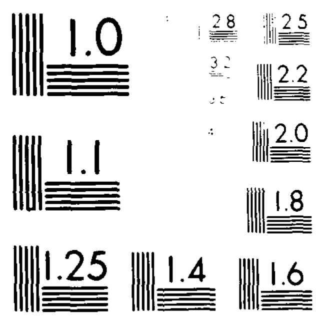
AD-A107 098   HONEYWELL SYSTEMS AND RESEARCH CENTER MINNEAPOLIS MN   F/G 14/5
COMPUTER IMAGE GENERATION: ADVANCED VISUAL/SENSOR SIMULATION.(U)
OCT 81   D SERREYN, D DUNCAN   F33615-80-C-0006
UNCLASSIFIED   AFHRL-TP-81-23   NL

1 OF 2
AD A
107098



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFHRL-TP-81-23

LEVEL II

(12)

AIR FORCE

HUMAN RESOURCES

AD A107098

COMPUTER IMAGE GENERATION: ADVANCED VISUAL/SENSOR SIMULATION

By

David Serreyn
David Duncan
Honeywell Systems and Research Center
2600 Ridgway Parkway
Minneapolis, Minnesota 55413

OPERATIONS TRAINING DIVISION
Williams Air Force Base, Arizona 85224

October 1981

Final Report

Approved for public release; distribution unlimited.

LABORATORY

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235

81 11 04 063

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

WILBURN O. CLARK
Contract Monitor

MILTON E. WOOD, Technical Director
Operations Training Division

RICHARD C. NEEDHAM, Colonel, USAF
Chief, Operations Training Division

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
AFHRL-TP-81-23

2. GOVT ACCESSION NO.
AD-A107098

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
COMPUTER IMAGE GENERATION: ADVANCED VISUAL/SENSOR SIMULATION

5. TYPE OF REPORT & PERIOD COVERED
Final

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
David Serrey
David Duncan

8. CONTRACT OR GRANT NUMBER(s)
F33615-80-C-0006

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Honeywell Systems and Research Center
2600 Ridgway Parkway, P.O. Box 312
Minneapolis, Minnesota 55413

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
62205F
61142109

11. CONTROLLING OFFICE NAME AND ADDRESS
HQ Air Force Human Resources Laboratory (AFSC)
Brooks Air Force Base, Texas 78235

12. REPORT DATE
October 1981

13. NUMBER OF PAGES
152

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Operations Training Division
Air Force Human Resources Laboratory
Williams Air Force Base, Arizona 85224

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

bicubic splines	fractals
computer image generation (CIG)	textured terrain
flying training	visual simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study investigated, developed, and evaluated various Computer Image Generation (CIG) techniques to overcome the qualitative and quantitative limitations of current CIG imagery produced by edge-based systems. The study concluded with an integration of techniques into a system concept. This report describes the techniques investigated, the system concept developed, and the general hardware implementations which are useful for cost/benefit tradeoffs. The system concept presented is based on the use of textured terrain for realistic simulation. In areas where the techniques were lacking or absent, a new approach was developed. The priority in choosing a technique was for training effectiveness and an algorithmically unified system. The approach also involves the display of terrain as curved surfaces represented by bicubic splines.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407341

AFHRL Technical Paper 81-23 October 1981

COMPUTER IMAGE GENERATION:
ADVANCED VISUAL/SENSOR SIMULATION

By

David Serreyn
David Duncan

Honeywell Systems and Research Center
2600 Ridgway Parkway
Minneapolis, Minnesota 55413

Reviewed by

Wilburn O. Clark
Contract Monitor

Submitted for Publication by

Warren E. Richeson
Chief, Engineering Branch
Operations Training Division

This publication is primarily a working paper.
It is published solely to document work performed.

PREFACE

This study was conducted by the Operations Training Division, Air Force Human Resources Laboratory, Air Force Systems Command. The study supports Project 6114, Simulation Techniques for Air Force Training, Warren E. Richeson, project monitor; and work unit 61142109, Advanced Computer Image Generation Concept Development, Dr. Wilburn O. Clark, work unit monitor. The principal investigator for this research was initially Mike Schroeder. He continued through the Literature Search, Detailed Study Plan, and Detailed Investigation of Candidate Techniques tasks. Then David Serreyn became the principal investigator for the System Concept and Benefit/Cost Analysis tasks. David Duncan was another significant contributor.

CONTENTS

CONTENTS (concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

LIST OF ACRONYMS

AVSS	Advanced CIG Visual/Sensor Simulation
CGI	Computer-Generated Imagery
CIG	Computer Image Generation
DLMS	Digital Land Mass System
DMA	Defense Mapping Agency
ECL	Emitter-Coupled Logic
FIFO	First In-First Out
FLIR	Forward-Looking Infrared
FOV	Field of View
HSV	Hue-Saturation-Value
IC	Integrated Circuit
IR	Infrared
LLLTV	Low Light Level Television
LSI	Large-Scale Integration
LUT	Look-Up Table
RAM	Random Access Memory
RGB	Red-Green-Blue
SMC	Surface Material Category
TTL	Transistor-Transistor Logic
VLIC	Very Large Integrated Circuit

SECTION 1

INTRODUCTION

The purpose of the Advanced Computer Image Generation (CIG) Visual/Sensor Simulation (AVSS) study contract was to investigate, develop, and evaluate various CIG techniques to overcome the qualitative and quantitative limitations of current CIG images produced by edge-based systems. The study was to conclude with an integration of these techniques into a system concept. This final report for the study phase of the contract discusses the investigation, development, and evaluation which led to the system concept.

This program is envisioned as part of a total system development where the next phase would consist of a software simulation/validation followed by a fabrication and testing of prototype hardware. The findings of the report, however, are directly applicable to many simulators and could be incorporated into other simulation systems. Emphasis on low-level flying and the need to incorporate tracking for target detection, recognition, and identification give rise to new requirements for realism in these simulations. Hue and saturation are important in long-range views. At medium and short ranges, texture gradients provide the depth cues needed for realistic simulation. This has been the primary basis for the system concept chosen for the study.

The criteria used in choosing the techniques have been the requirements for:

- Capability to provide more realism than current techniques
- Potential for greater efficiency than current techniques
- Compatibility with other techniques from a system point of view
- Reasonable architecture for using today's technology
- Possibilities for reduction in cost using tomorrow's technology

The major areas of concern in CIG for visual/sensor simulation are:

- Surface representation
- Texture representation
- Intensity computation
- Image generation
- Embedded objects

The method used for perspective views of a surface is that of using bicubic subdivision originally proposed by Catmull (1974). Catmull's method is a fast recursive subdivision of a patch into smaller and smaller subpatches until each subpatch is small enough to be displayed; that is, pixel size.

The texture representation in this study is based on fractals which are especially suitable for training in low-level flight and for air-to-surface missions. The implementation of fractal texturing allows computing the surface to arbitrary levels of detail without increasing the data base. Therefore, the simulation would result in a continuous change of detail and texture density for all flight paths.

The intensity calculation is based on color--hue, saturation, and value (HSV)--for the visible. The hue, saturation, and value are converted to red, green, and blue (RGB) for display purposes. For low light level TV (LLLTV) and forward-looking infrared (FLIR) black and white will be used (same as in the actual sensor). There are many more sensor requirements, however, when simulating. The parameters which affect the value or intensity in the display must be determined prior to the actual simulation.

The image generation is based on a z-buffer concept. Those surfaces which are closer occlude all other surfaces. This is a simple solution to the hidden surface problem. Computing requirements increase linearly with scene complexity. Also, different atmospheric effects are possible since the distance to each pixel is known. Moving targets and objects within the scene can be generated simultaneously in this approach.

This report covers these aspects of the technical effort of the study phase for AVSS. The system concept (Section 2) is presented first, followed by the detailed investigation (Section 3). Some implementation considerations are discussed in Section 4. The conclusion (Section 5) precedes a list of references used in the investigation for this report. Appendixes A through E are included to provide additional detail about some of the concepts investigated.

SECTION 2

SYSTEM CONCEPT

The general system concept is presented in this section. In Section 3, more detail is given where appropriate to complete the system concept.

The system concept developed for AVSS is shown in Figure 1. The system consists of both off-line (computer-generated) preparation and on-line or real-time generation. The off-line preparation consists of obtaining mission inputs from the Defense Mapping Agency (DMA) data base as well as additional mission information such as time of day and sensor. Once this information is obtained, the appropriate files are merged into one large file for further processing. Texture labels are generated for each of the terrain squares. Note that the texture labels and register squares are mission-independent, and can be generated any time prior to the mission. Next shadows are generated and stored on the disc drives. Finally, cultural features such as houses, bridges, and roads are added.

The on-line generation consists of determining the initial starting point and loading the fast field-of-view (FOV) memory. Once this is done, only new FOV information must be transmitted between the on-line storage and the fast FOV memory. A perspective transformation then transforms the data from object space to image space. This requires a test of the patch size to see whether it is also pixel size. If it is not, it is subdivided until it is equal to a pixel size.

Subdivision involves more than just subdividing surface polygons down to pixel size. The fractal information being used for intensity calculation must be subdivided and the surface normals for each subpatch must also be calculated.

Intensity calculation must now be done for every pixel within the whole screen. However, some pixels may be overridden because they are hidden; it is this occultation that is of concern. As each pixel's intensity is calculated, the z-buffer information (the distance information) is checked to see whether it is closer to the pilot than what is already contained within the memory. If the z-buffer information is closer to the pilot's position, it will overlay the memory data. Once the intensity is calculated an anti-alias procedure removes the jagged edges in the scene. Once anti-aliased, the data is dumped into a frame buffer and displayed. There should be two frame buffers so that one can be filled during the time that the second is being displayed.

A 1024 x 1024 display and a 60-Hz refresh rate are ambitious assumptions; each of the blocks just discussed will be examined in more detail.

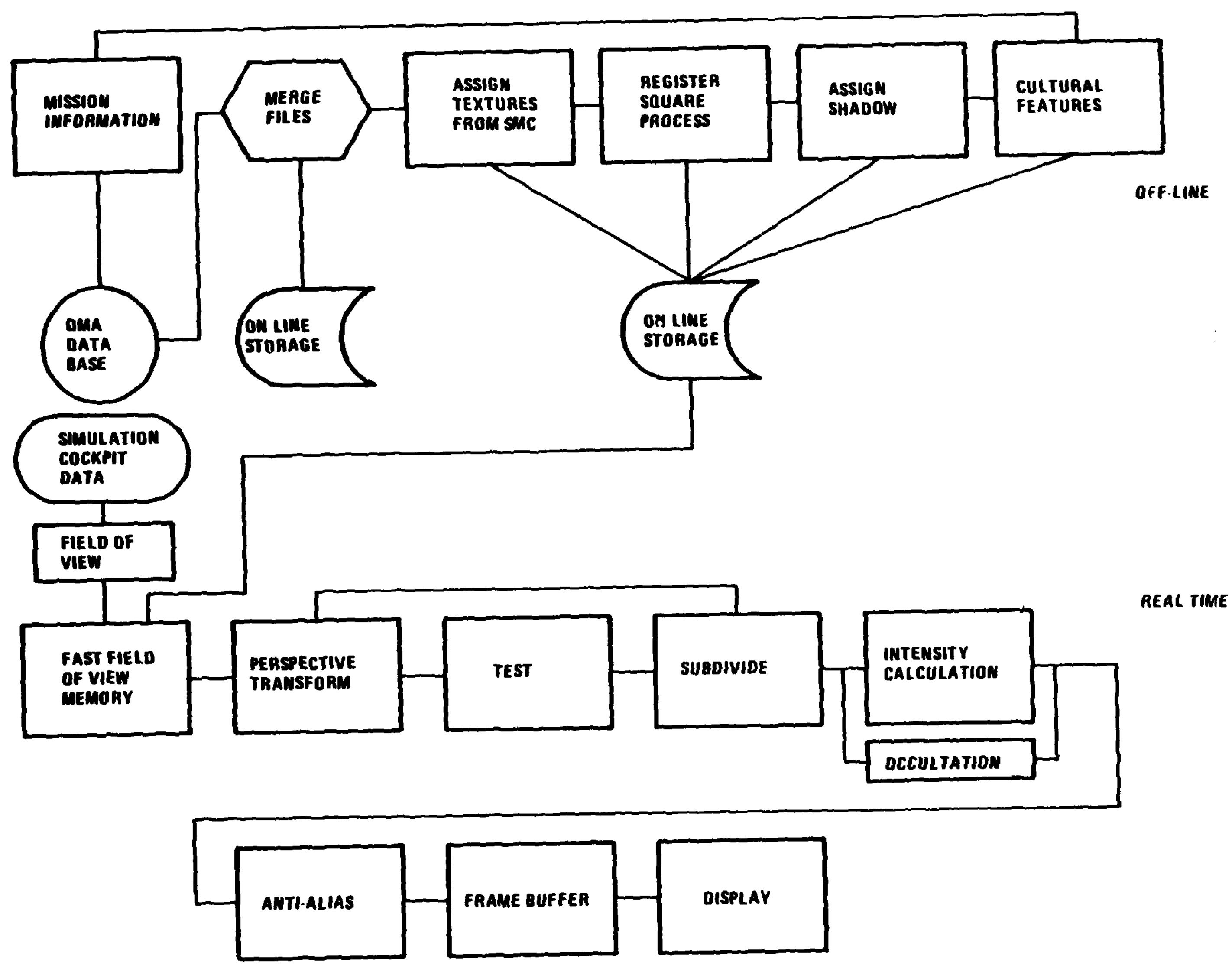


Figure 1. AVSS system concept.

The following mission information is assumed to be typical of the type to be given:

- o Sensor type: visual, infrared (IR), or LLLTV
- o All the weather conditions for the duration of the entire flight. (Assuming one constant set of weather conditions.)
- o The sun or the moon position. (For the entire duration of the mission the sun and moon position are considered constant.)
- o Albedos (reflectivities) and emissivity information
- o Special effects
- o Additional cultural objects
- o Fractal look-up tables (LUTs) to be designated for texture, texture borders, and shadow borders.
- o The approximate flight path which gives us the required inputs for the DMA data base size and location.

The first job is to select the appropriate DMA data files, which consist of terrain elevations as well as cultural data. These data are then merged so that all the elevations are in one file and the cultural data are coordinated with that same file. This particular operation of reading the tapes could require several hours because up to 10 tapes may have to be searched to obtain all the required data for an assumed 800 km x 48 km flight path area. These elevation and cultural data require about 10 megabytes (Mbyte) for storage.

The next procedure assigns the textures in the surface material categories to the DMA data base and then locates the particular elevations in that texture area. The first step is to construct a chain code and then to locate and designate the texture labels.

The register square process computes the register square values for each patch corner. To do this, 16 elevations are placed in a 4 x 4 matrix to get the register square values. The method used is a bicubic spline which allows for no discontinuity between patches. The time required for this particular operation is about four hours for the entire mission data base.

Next, shadows are assigned so that each terrain square has a shadow label. The procedure is first to determine the pixel dimensions that are contingent on the terrain elevation, then to determine the sun space FOV, and finally to determine the z values of the distance information. Now those areas that are shaded can be determined and labeled. This is similar to the register square process and will take approximately 13 hours.

Finally, the cultural features desired for a particular mission are added. These features include houses, trees, roads, railroads, factories, lights, moving objects, etc.

After all these off-line operations, the result is the mission data base. It is a collection of all the regional data blocks (or patches) contained in the mission data base. The regional data blocks contain register squares defining the bicubic elevations, a pointer to a list of the unique features which apply to the patch, a pointer for the synthetic texture, a definition and list of objects, and a surface normal. The data blocks should be organized as a quad tree to facilitate finding the patches for a given frame.

The next step involves the real-time system requirements for AVSS. Initially, some information about the starting location of the aircraft must be assumed, and these data must be transferred from the on-line storage to the fast FOV memory. As the FOV changes, only the resulting new data must be transferred from the on-line storage to the fast FOV memory. The data storage for the fast FOV memory is expected to be about 12 Mbytes for a full 360° FOV. This is based on having approximately 368 bits for each patch, consisting of the vector normal, xy location, size, texture, shadow, and register square data.

The next three blocks--transformation, test, and subdivide--are generally treated as one unit. A patch is put into the perspective transformation and transferred from object space to image space. The patch is then tested to see whether it is of pixel size. If not, it must be subdivided in object space and undergo another perspective transformation into image space, and then be tested again. This procedure involves a large number of perspective transformations per frame. Later sections will describe the number of perspective transformations and how the perspective and subdivision processes are performed. As discussed previously, subdivision consists of three parts: bicubic, fractal, and vector normal.

At the end of subdivision, approximately five million patches will require an intensity calculation. Each must have vector normalization and must be transformed from HSV into the RGB space of the display. Only those polygons which are not hidden will go through the anti-aliasing; hence, the occultation is really a test to determine which parts are hidden. The purpose of anti-aliasing is to eliminate aliasing artifacts caused by improper sampling. Our procedure is to have a five-point average pixel that uses three points per frame. This requires three times the number of pixels in the frame operations.

The frame buffer and display can be considered as a unit. The frame buffer feeds the display; however, it gets the data from the rest of the hardware. Two buffers are used so that one can be inputting data at the same time that the other is displaying. The initial concept involves 22 Mbytes of high-speed frame buffer memory, for both intensity and z-buffer information. How this might be eliminated later is discussed in succeeding sections. The intensity buffer can be preset to a given condition at the start of each frame. When the designated mission area has been ventured outside of during simulation, this preset condition will then occur.

SECTION 3

DETAILED INVESTIGATION

This section of the report presents a detailed investigation of the AVSS techniques. Included is a discussion of some alternatives considered, but most of the section focuses on the preferred approach.

3.1 OPERATIONAL REQUIREMENTS

Eight areas are to be considered under the category of operational requirements:

1. Update rate
2. Transport delay
3. Perspective
4. Viewpoint maneuverability
5. Field of view
6. Resolution
7. Image mapping
8. Occultation

3.1.1 Update Rate

In this study of the AVSS system, the update rate has been a fixed parameter which determines the complexity of the system before the frame buffer. The update has been chosen to be 1/60 second per frame; a frame is defined as 1024 pixels per line and 1024 lines per frame.

Even though a 1/30-second frame rate is acceptable for normal television video, some aliasing artifacts can still be seen, for example, a wheel turning backwards on a stagecoach. Note that neither more samples per line, nor an increase in frame rate, necessarily eliminate this phenomenon. Increasing the frame rate simply reduces temporal aliasing artifacts. In general, the update rate is sufficient to allow the viewer to perceive continuous motion and to interact with the simulated environment.

3.1.2 Transport Delay

Transport delay is defined as the interval from the time the aircraft state is calculated to the completion of the image computed for that state. The transport delay is the sum of several delays through the system:

$$T_D = T_{TRANSFORM} + T_{TEST} + T_{SUB} + T_{INT} + T_{AA} + T_{FB}$$

where

T_D = transport delay

$T_{TRANSFORM}$ = delay through the perspective transformation

T_{TEST} = delay through the test before subdivision

T_{SUB} = delay through the patch subdivision

T_{INT} = delay through the intensity calculation and z-buffer test

T_{AA} = delay through the anti-aliasing

T_{FB} = delay through the frame buffer

The approach taken here to estimate the delay was to assume fully parallel computation and to use pipeline architecture ideas for the individual clocks shown in the system concept. The throughput is determined by the lowest-speed device in the pipeline and should not be confused with the transport delay calculations. Each of the following estimates is based on these assumptions.

The first portion of the transport delay involves the time required to do the perspective transform from object space to image space. This projection requires five stages, as shown in Figure 2. (Note: Many of the figures and discussions are based on Solano, Voth, and Narendra; 1981). If 200 nanoseconds (nsec) per stage is assumed, then

$T_{TRANSFORM}$ = 1.0 microseconds (μsec)

It should be noted that the 200 nsec is really a clock time determined by the slowest element in the pipeline; in this case, the multiply. It may not be necessary to use 200 nsec for all stages.

The delay through the test consists of a box gridpoint test, which determines whether the patch should be further subdivided or passed through to the intensity calculation. A box approximation test is used; the implementation is shown in Figures 3 and 4. There are a total of eight stages at 200 nsec per stage, therefore,

T_{TEST} = 1.6 μsec

The subdivision process involves a total delay of four stages (Figures 5 and 6); the letters shown in the figures refer to the values in the register squares before and after subdivision. There is, however, a potential bottleneck at this location. When the subdivision occurs, four new sets of register square values are created. Unless one has a means of immediately continuing with four parallel channels, the data must be stored as shown in Figure 7. The delay through the memory must be included in the transport delay. Figure 8 presents a possible parallel structure for the total process of perspective transformation, testing, and subdivision.

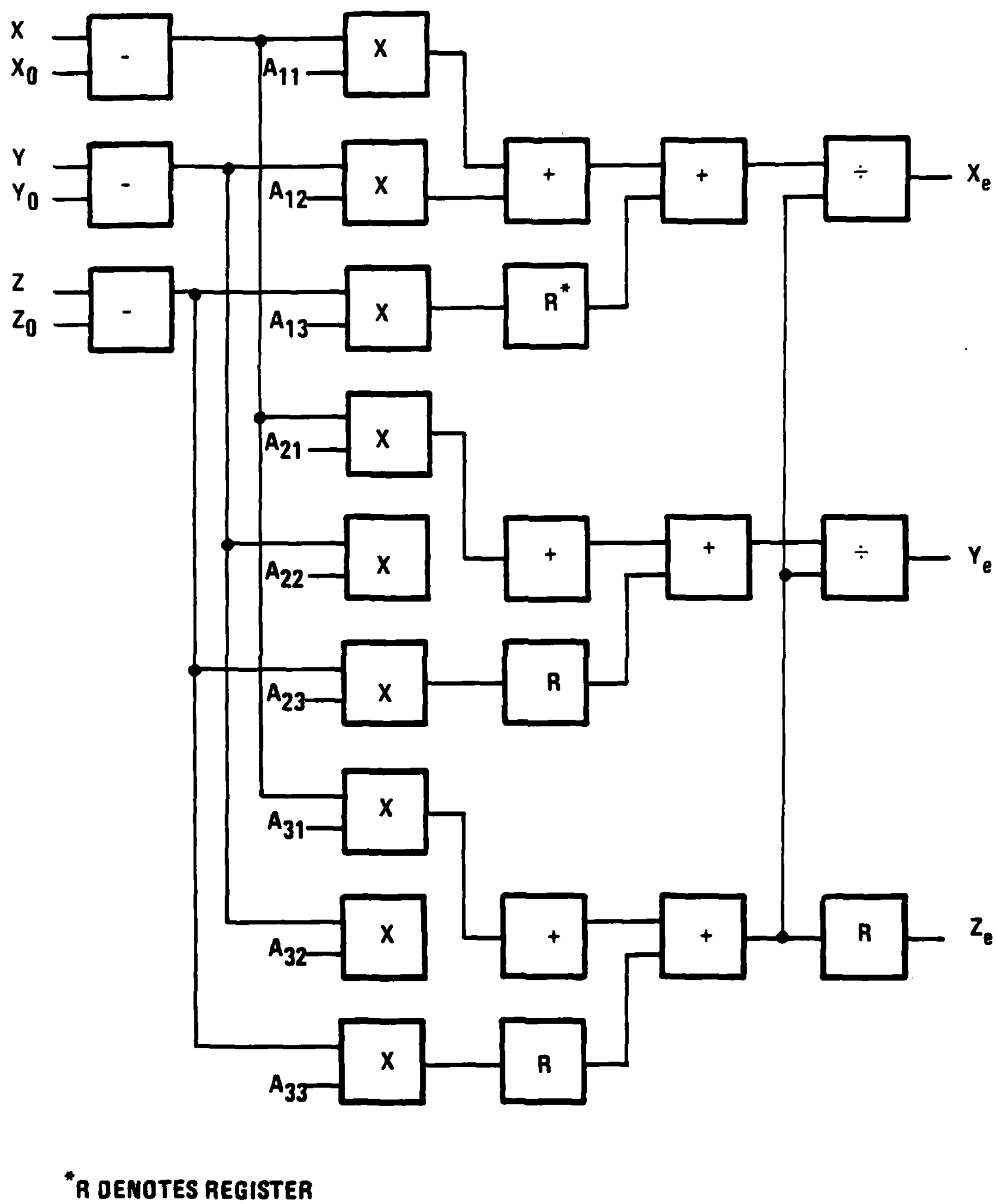


Figure 2. Pipeline hardware block diagram for perspective projection stage.

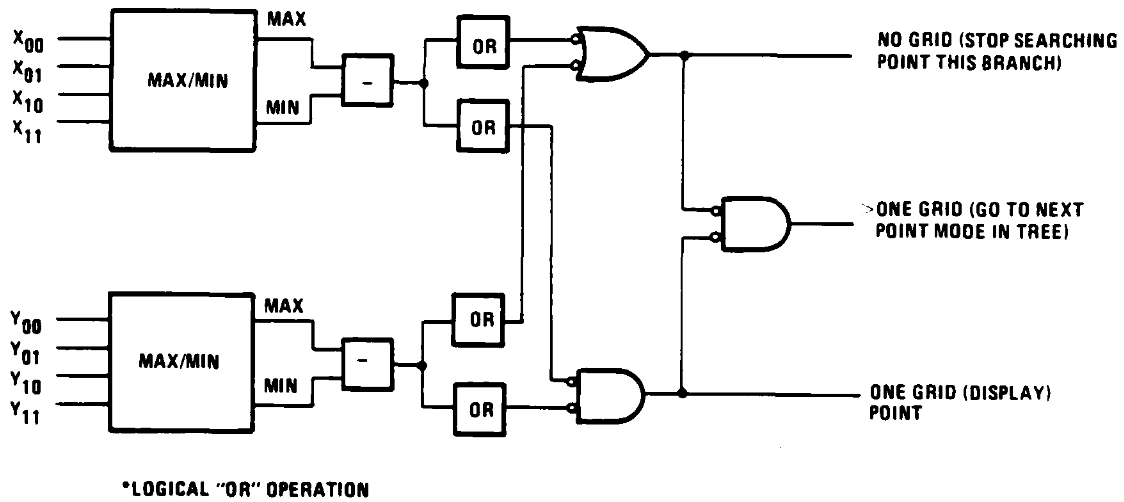


Figure 3. Grid point test.

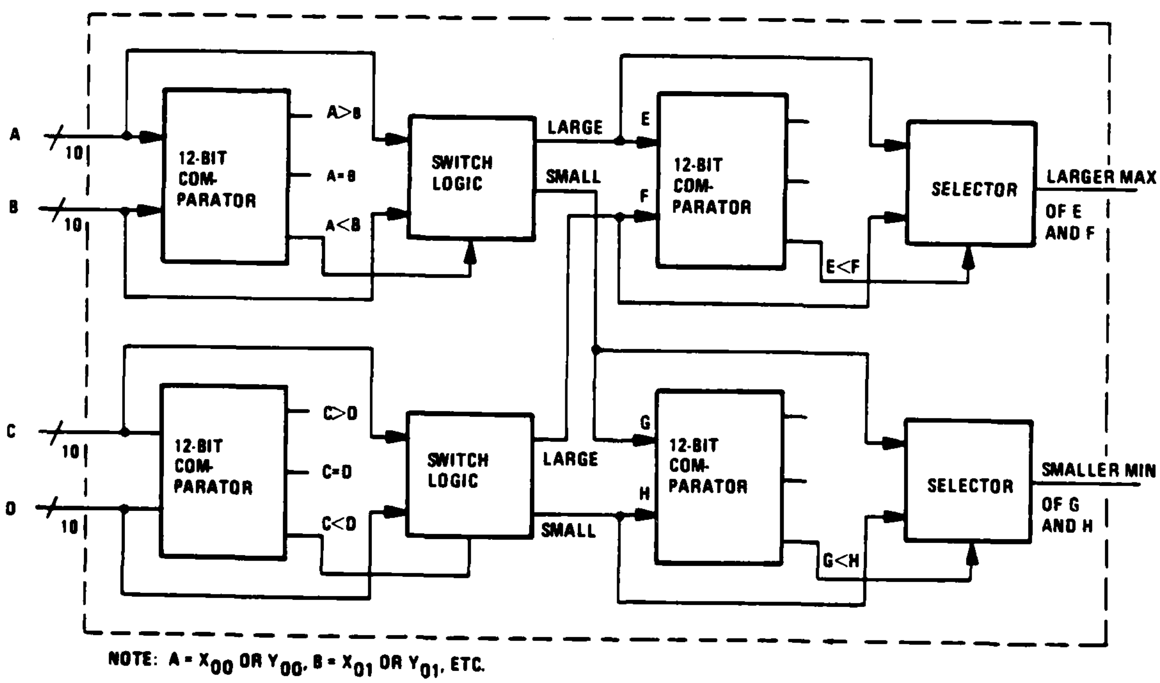
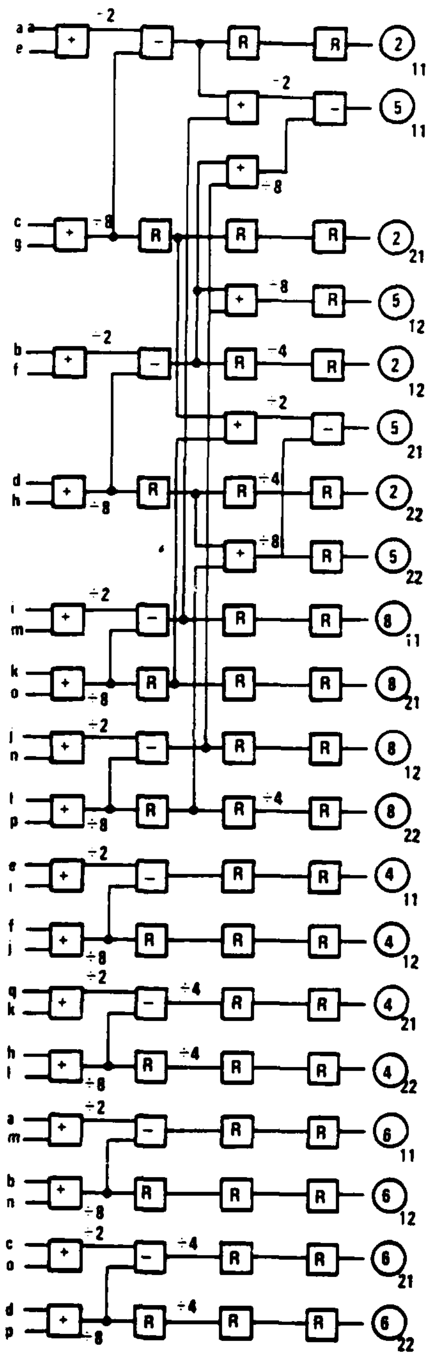


Figure 4. Maximum/minimum circuit as part of grid point test.

SUBDIVIDE PATCH (PART I)



NOTE: SUBSCRIPT IDENTIFIES ONE OF FOUR NEW REGISTER SQUARE VALUES IN ONE OF NINE NEW REGISTER SQUARES. DENOTED BY CIRCLED NUMBER

Figure 5. Subdivide pipeline, Part 1.

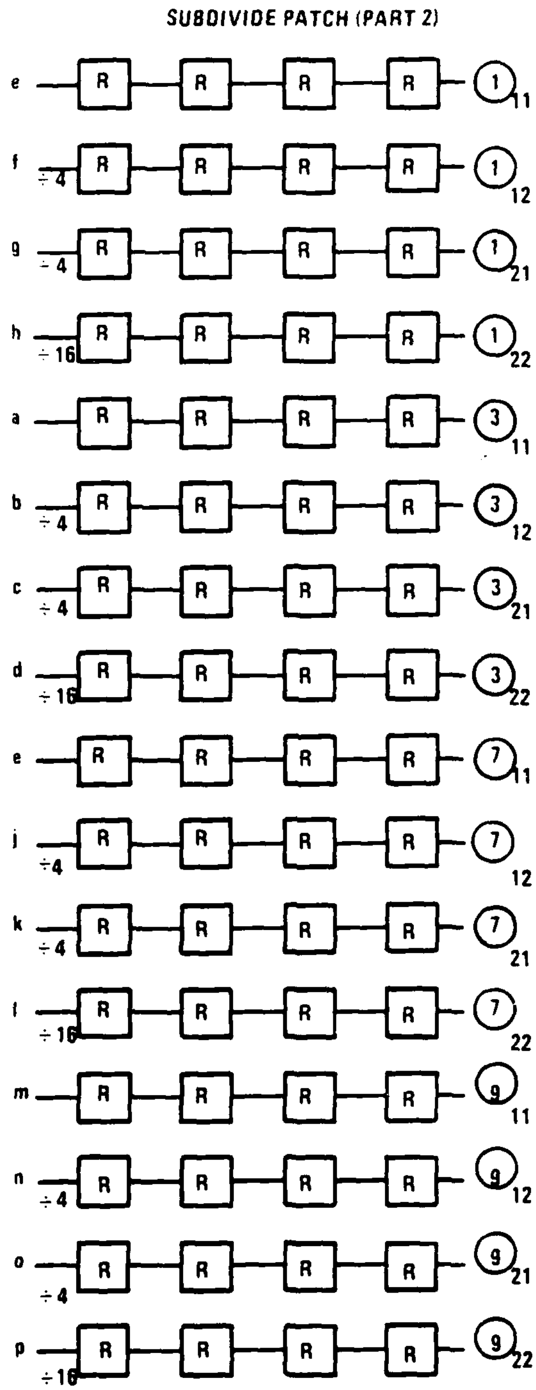


Figure 6. Subdivide pipeline, Part 2.

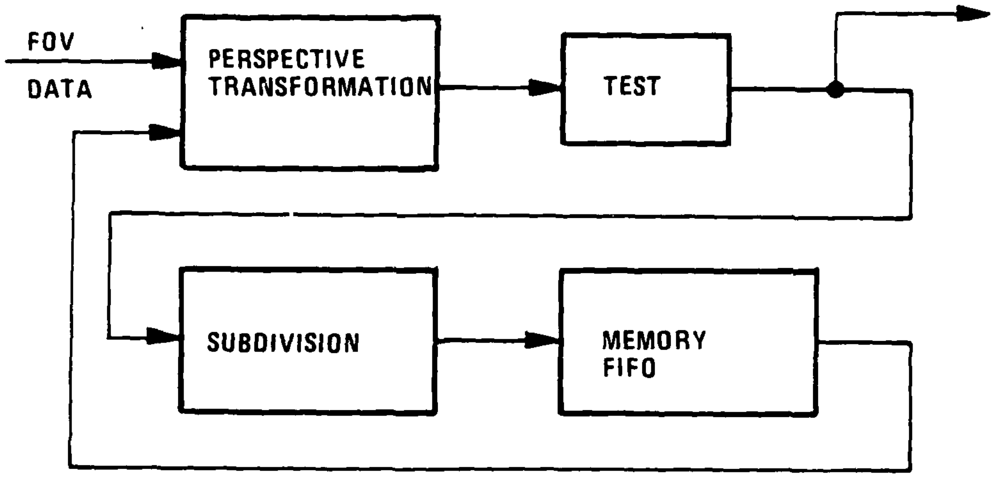


Figure 7. Subdivision process involving memory.

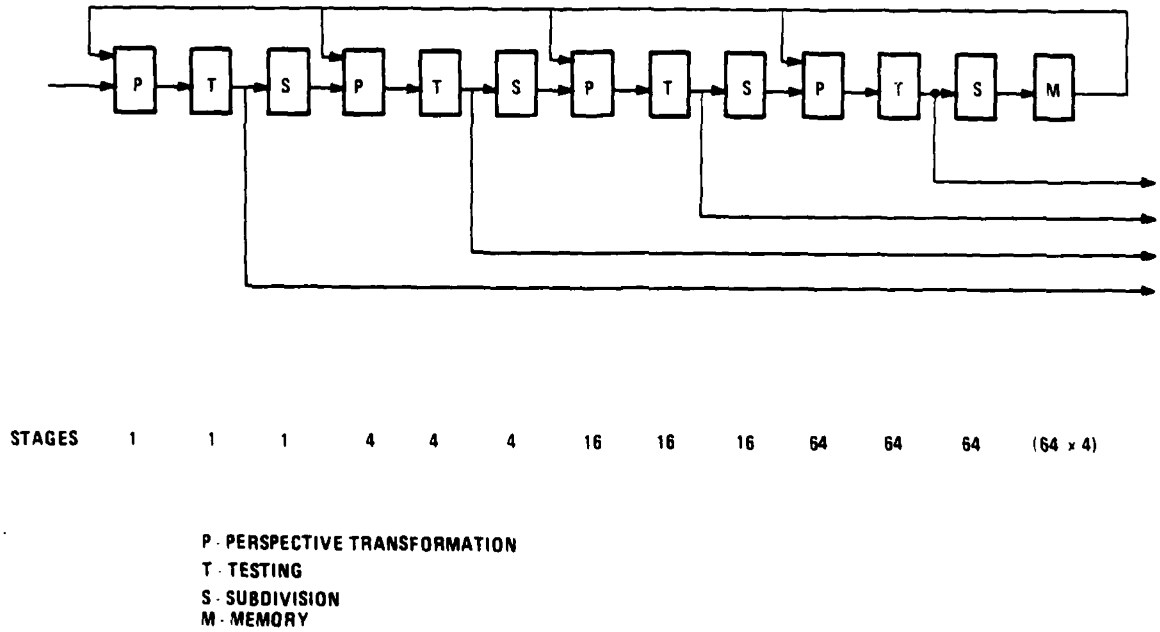


Figure 8. Perspective transformation involving loop process.

At the end of each test, data will either go to the intensity calculation or require further subdivision. The number of subdivision is shown to be about 1.7×10^6 in Appendix E, with the total number of patches to be four times this, or 6.8×10^6. For the worst case it is assumed that all this would have to be stored in the memory channels. Then the memory required for the channels shown is:

$$6.8 \times 10^6/64 \approx 106\text{K/channel}$$

Again assuming 200 nsec for the delay or clock cycle, the result is 0.021 seconds of delay, or 21 milliseconds (msec).

Note here that number of channels of the perspective transformation, test, and subdivision are each 85 channels. In Section 4, Implementation Considerations, it is estimated that approximately 80 channels are required to complete the entire process in 1/60 sec.

In this approach, transport delay in memory has been sacrificed for hardware. It is possible, for instance, to have 256 stages of additional perspective transform and test. The delay would go from milliseconds to microseconds.

From the above discussions:

$$T_{FIFO} = 21 \text{ msec}$$

and

$$T_{PROC} = 0.0008 \text{ msec (which is neglible)}$$

giving

$$T_{SUB} = 21.0008 \text{ msec}$$

Note in Figure 8 that the output of the First In-First Out (FIFO) stack or memory is going to several of the perspective transformations. If no data are going into the perspective transformation at that stage, then that stage and testing can be taken advantage of. The analysis above has not considered this specifically. Also, the ramifications of all this feedback must be taken into account in the controlling. It may be that the memory could be allocated so that the transport delay could be reduced further with no increase in memory but with more control circuitry.

The intensity calculation has an estimated 22 stages. This is the estimated complexity for the equations given in the tonal computation section. The implementation block diagram is shown in Figure 9. Assuming 200 nsec per stage, then:

$$T_{INT} = 4.4 \ \mu\text{sec}$$

The anti-aliasing delay comes from four stages of multiply, two adds, and switch between frames. Hence:

$$T_{AA} = 0.8 \ \mu\text{sec}$$

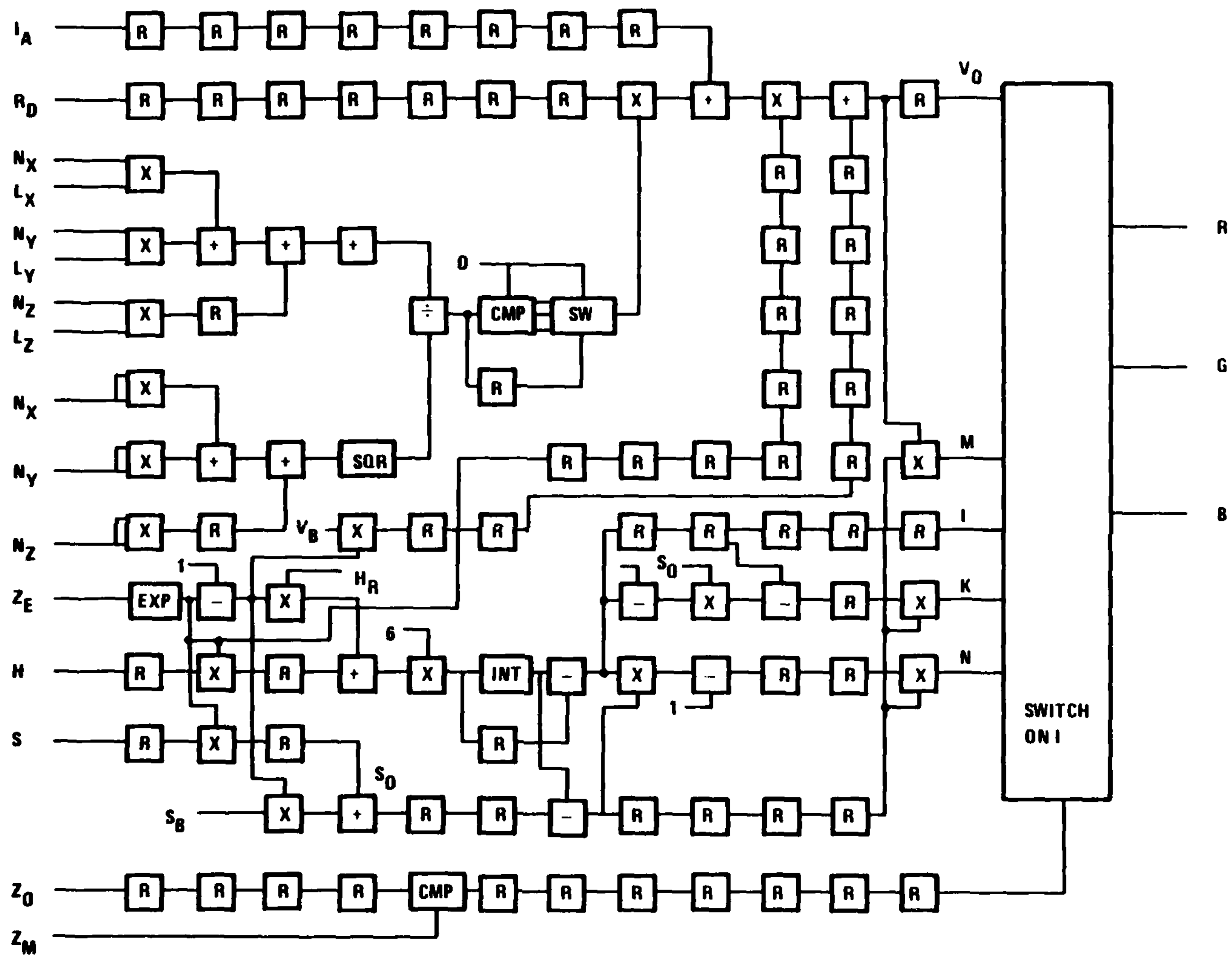


Figure 9. Intensity calculation and occultation.

The system concept approach involves switching between two buffers. While one frame is being calculated, the second is being displayed. The maximum delay is one frame time, or

$$T_{FB} = 16.6667 \text{ msec}$$

The total delay time is

$$\begin{aligned} T_{TRANSPORT} &= 0.001 + 0.0016 + 21.0008 + 0.0044 + 0.0008 + 16.6667 \\ &= 37.6753 \text{ msec} \end{aligned}$$

One item not included in the above transport delay is the time to get data from the disc storage to the fast FOV memory. If a 360^{0} fast FOV memory is assumed, then the only data to be added to the data base are those data corresponding to the distance traveled in level flight. It is shown in subsection 3.1.4 that 10 patches need to be transferred to high-speed memory. These 10 patches are at the extreme viewing distance so that when the new patches actually appear, they will not be noticeable. The primary concern is the seek, latency, and transmit time of the disc; that is:

$$T_{DISC} = T_{SEEK} + T_{LATENCY} + T_{TRANSMIT}$$

For a typical mass storage unit, the seek time is 30 msec (average), the latency time is 8.33 msec (average), and for 10 patches at 1.5 Mbyte transfer rate, the transmit time is 0.3 msec, which gives:

$$T_{DISC} = 38.63 \text{ msec (average)}$$

When this is added to the previous time, the total transport time is:

$$T_{TRANSPORT} \simeq 76 \text{ msec}$$

This time can be considerably decreased by getting patches only every tenth frame, which reduces the average disc transport delay to about 4 msec per frame.

3.1.3 Perspective

In order to preserve perspective validity in the displayed imagery, only two kinds of transformations have been considered: one to a flat screen, the other to a spherically curved screen. For reasons discussed in subsection 3.1.7, the flat screen transformation was chosen.

Physically, the perspective transformation involves three coordinate systems (Figure 10): 1) the world coordinate system, in which all off-line and on-line terrain and cultural features are initially located; 2) the eye coordinate system, a rectangular orthogonal system with its origin at the pilots' eye; and 3) the screen coordinate systems, representing the location of a real screen or window (see Newman and Sproull, 1979, for details).

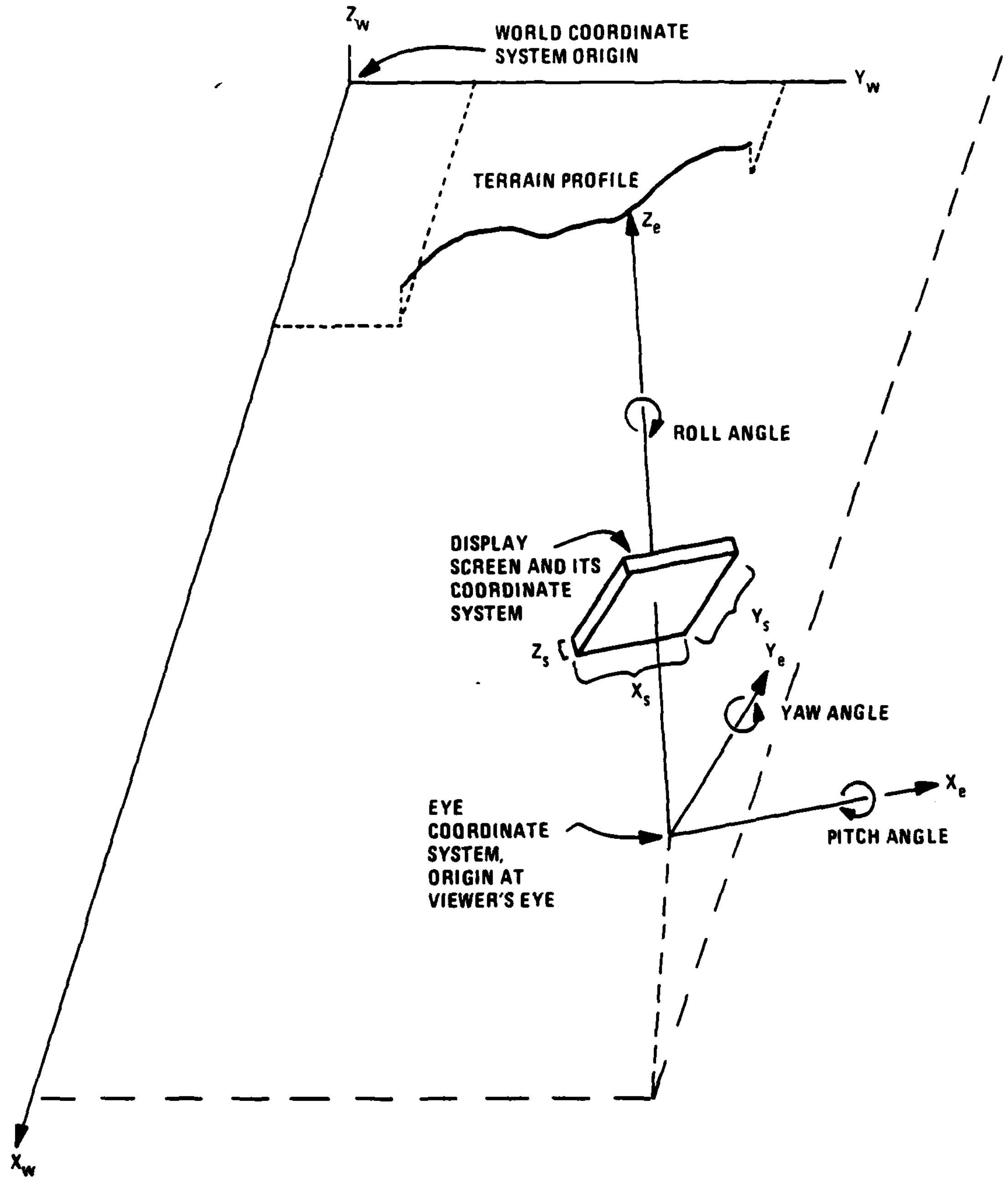


Figure 10. Coordinate systems.

The purpose of the transformation is to take many individual three-dimensional points in world coordinates and transform them to screen coordinates or, equivalently, screen space. The eye coordinate system serves as an intermediary between world and screen space. The entire transformation of an arbitrary point from world eye to screen space is symbolically

$$(X_w, Y_w, Z_w) \quad (X_e, Y_e, Z_e) \quad (X_s, Y_s, Z_s)$$

Thus, two transformations are involved.

The first, from world to eye coordinates, involves a 4 x 4 matrix multiply

$$(X_e \; Y_e \; Z_e \; 1) = (X_w \; Y_w \; Z_w \; 1)V$$

where V is a 4 x 4 matrix concatenated from five component matrixes:

$$V = V_1 \cdot V_2 \cdot V_3 \cdot V_4 \cdot V_5$$

The five component matrixes are

1. Translation matrix:

$$V_1 = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -t_x & -t_y & -t_z & 1 \end{matrix}$$

2. Right-handed to left-handed matrix:

$$V_2 = \begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

3. Rotate by Θ around y_e axis:

$$V_3 = \begin{matrix} \cos\Theta & 0 & \sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

4. Rotate by ϕ around x_e axis:

$$V_4 = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

5. Rotate by ψ around z_e axis:

$$V_5 = \begin{matrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Angles are measured clockwise when looking along the rotation axis toward the origin. The multiplication of the V matrixes gives the A_{ij} values presented in Figure 2.

Following the convention of many computer graphics people, the eye system is left-handed, X_e to the right, Z_e into the screen, Y_e up. Because the world system is right-handed, the matrix V_2 is employed to change from right to left. Of course, this convention may be ignored and V_2 excluded at the convenience of the training system designer.

The eye coordinate system has its origin at t_x, t_y, t_e in the world system. The eye axes are rotated through three angles: Θ, ϕ, ψ, (yaw, pitch, and roll, respectively). Measuring angles as shown in Figure 10 with world and eye axes initially aligned, Θ, ϕ, ψ are respectively the rotation angles about the Y_e, X_e, Z_e axes. Note that these rotations must occur in the order given to make sense; that is, the rotation matrixes do not commute.

This viewpoint transformation allows presentation of the proper scene for any viewpoint within the defined data base. The viewpoint can be from any altitude, including on the surface. The attitude (roll, pitch, and yaw) is not restricted. The sine and cosine terms must be calculated, however. This calculation is done once per frame; that is, it will not change during the frame time.

The second part of the perspective transformation $(X_e\ Y_e\ Z_e) \rightarrow (X_s\ Y_s\ Z_s)$ is given by

$$X_s = \frac{DX_e}{SZ_e} V_{sx} + V_{cx}$$

$$Y_s = \frac{DY_e}{SZ_e} V_{sy} + V_{cy}$$

and Z_s may be either

$$Z_s = \frac{S((Z_e/D) - 1)}{(1 - D/F)\ (SZ_e/D)} \quad \text{or } Z_s = Z_e$$

X_s and Y_s are orthogonal screen coordinates, D is the screen center-eye distance and S is half the screen size in the same units as D. (The screen is assumed square.)

The four V variables are windowing parameters. The window (screen) is $2V_{sx}$ units wide and $2V_{sy}$ units high and in the same units; the screen center is at (V_{cx}, V_{cy}).

Depth (Z_s) is required for occultation only. The simplest form for Z_s is $Z_s = Z_e$. This form is quick because it avoids extra computation; however, a large number of bits may be required for its perceptually adequate application in occultation tests. The other form for Z_s is computationally expensive but allows for finer depth resolution nearby and coarser resolution in the distance, so that fewer bits are required for the specification of Z_s. In this form, "F" is the maximum depth allowed in the system. Note that as F approaches , $Z_s = 1 - D/Z_e$, demonstrating the relationship between resolution and distance more clearly. The form $Z_s = Z_e$ should be used for real-time application.

The benefit from this portion of the perspective transformation is that the scene will be displayed as if seen or sensed from an actual aircraft at an identical position and orientation. The major concern is the division by the object distance. This can be time-consuming unless one uses a fast multiply chip and a reciprocal for the number. Fast divide chips are expected to be available soon.

3.1.3.1 <u>Operations</u>--The perspective transformation of a patch is accomplished by transforming all four patch vertexes in parallel. Since one subdivision produces four patches, the total number of perspective transformations per frame time equals four times the number of subdivisions per frame time. Solano, Voth, and Narendra (1981) estimate that, on the average, rough terrain requires 1.7 x 10^6 subdivisions per frame. Consequently, 6.8 x 10^6 perspective transformations/frame are expected on the average. Each vertex transformation requires nine multiplies, six adds, and two divides. The multiplication and divide operations can be implemented in parallel, as shown in Figure 2. Also the adds are both in series and in parallel, as shown in Figure 2.

3.1.4 <u>Viewpoint Maneuverability</u>

A viewpoint may be defined by six variables: three for spatial position and three for orientation. The Honeywell algorithm allows any value for all of these variables and so accommodates all possible viewpoints. The speed with which a viewpoint may change is limited by the access time of obtaining new patch information. There are essentially three types of movement which require new patches:

1. Roll--longitudinal axis of aircraft constant
2. Movement in space--orientation constant
3. Pitch and yaw--change in orientation

For horizontal movement in space, a small circular crescent-shaped area must be obtained each frame time. The area is given by:

$$A = hR$$

where

h = distance flown horizontally (worst case)

R = radius to the horizon

with a velocity of 500 mph or 223.5 m/sec, h = 3.725m and R = 24 km gives A = 90,000 m^2 = 10 patches * 368 bits/frame time.

The angular velocity of roll is proportional to flight speed. Under extreme conditions, a pilot may perform a 360° roll in 1 sec. This is equivalent to 6° in one frame time. The front window FOV will not change much with a roll, but side window FOVs will change considerably. For a 60° x 60° side FOV, a 6° change is the same as 1/10 of the FOV. With the side view inclusive of all patches out to the horizon within a 60° sweep, changing the screen FOV by 1/10 will at worst introduce a few nearby patches during horizontal flight. For a vertical flight path, a roll will require updating 1/10 x 40,000 patches each frame time, which we regard as the worst-case roll:

Worst Case Roll = 4000 patches x 368 bits/frame time

When the pilot changes the aircraft direction, he is limited by the inertial forces on himself and the aircraft. We shall assume a maximum acceleration of 9 g's. A constant change in direction at constant speed results in a circular path with a centripetal acceleration of:

$$9g\text{'s} = \frac{v^2}{R}$$

where

v = forward aircraft speed

R = radius of aircraft circular path

with v = 223.5 m/sec, and 1g = 10 m/sec^2, R = 555m. The circle circumference = $2\pi R$ = 3487m.

In 1/60 sec, the aircraft travels d = vt = 223.5/60 = 3.725m and the change in the aircraft heading "Δ" is given by:

$$\frac{\Delta}{360^\circ} = \frac{3.725}{3487} \; ; \; \Delta = 0.38^\circ$$

Turning by 0.38° requires updating 253 patches x 368 bits each/frame time. Thus the transfer rate of each case is:

1. 27 Kbytes/sec
2. 11 Mbytes/sec
3. 0.7 Mbytes/sec

The second and third cases have assumed a transfer rate from disc to high-speed memory necessitated because the high-speed memory only covers the FOV of 60°. If the high-speed memory is increased to hold all 360° out to 24 km, cases 2 and 3 are no problem, and case 1 becomes the limiting transfer rate. To avoid incoherent imagery during rolls, the larger high-speed memory is considered to be the superior option.

3.1.5 Field of View

The concept developed in this study can generate valid images independent of the FOV and can simultaneously compute images for multiple image planes. The standard FOV for this report is 60°; however, any FOV from 0° to 180° may theoretically be employed for each image plane. The complexity of this system increases with the FOV; that is, 180° requires approximately three times the hardware of 60°.

There are two parts to the solution for selecting the FOV. The first part requires traversing a data structure called a quad tree. The quad tree, at its highest level, divides the data base into quarters. At the next level down the tree, each quarter is itself subdivided into quarters. This subdivision continues until the smallest resolution in the data base is read. This is the level at which the subdivision dimensions match the sample interval between elevations in the world data base. The areas of data corresponding to the smallest data base resolution are called patches.

The first part of the FOV selection requires collecting all patches contained in the FOV. This is done by projecting levels of the quad tree to the display coordinates. If a subdivision lies entirely outside the viewing window, that subdivision and all those lower than it can be pruned from the tree. Patches partially in the FOV should be subdivided by the method described in subsection 3.2.3 and then those subpatches should be tested for inclusion.

After all patches wholly or partially contained in the FOV have been located, the parts of polygonal objects outside the field need to be clipped out. That is, some objects may be partially in the FOV. So as not to waste time processing points that are not to be displayed, these objects are clipped at the borders of the FOV.

3.1.6 Resolution

A 60° FOV on a screen of dimensions 1024 x 1024 pixels is assumed. This is one-sixth of the data we have stored in the fast FOV memory. For the vast majority of cases, this is sufficient for sharp, clear imagery. A further increase in apparent resolution is obtained by the anti-aliasing method. Leler (1980) describes this perceptual effect in some detail. The image generation system has been designed to meet visual resolution standards and is more than adequate for the less resolute IR and LLLTV sensor simulations.

3.1.7 Image Mapping

This subsection discusses the applicability of a spherical display surface designed to achieve a constant resolution over the entire display. Compared to a flat display surface, a spherical surface can ostensibly reduce the system cost by requiring fewer total pixels.

Consider the three coordinate systems illustrated in Figure 11; (X_w, Y_w, Z_w) are the stationary world coordinates, (X, Y, Z) are "parallel" coordinates with axes always parallel to world coordinates and with origin at the pilot's eye, and (X_e, Y_e, Z_e) are orthogonal coordinates with the same origin, but with axes oriented so that Z_e pierces the screen center and X_e and Y_e are, respectively, parallel to the vertical and horizontal screen borders. The eye coordinates and screen are detailed in Figure 12. Note that screen coordinates are given by the two angles Θ and ϕ.

Given a point (P) in world coordinates, its parallel coordinates are

$$X = X_w - O_x \quad Y = Y_w - O_y \quad Z = Z_w - O_z$$

where O_x, O_y, O_z = origin in world coordinates.

In eye coordinates, (P) is given by

$$\begin{aligned} X_e &= (X_e,X)X + (X_e,Y)Y + (X_e,Z)Z \\ Y_e &= (Y_e,X)X + (Y_e,Y)Y + (Y_e,Z)Z \\ Z_e &= (Z_e,X)X + (Z_e,Y)Y + (Z_e,Z)Z \end{aligned}$$

where (X_e,X) = cosine of the angle between the X_e and X axes, (X_e,Y) = etc. Note that these direction cosines are constants over the entire frame, independent of the value of P, and are recomputed each frame time.

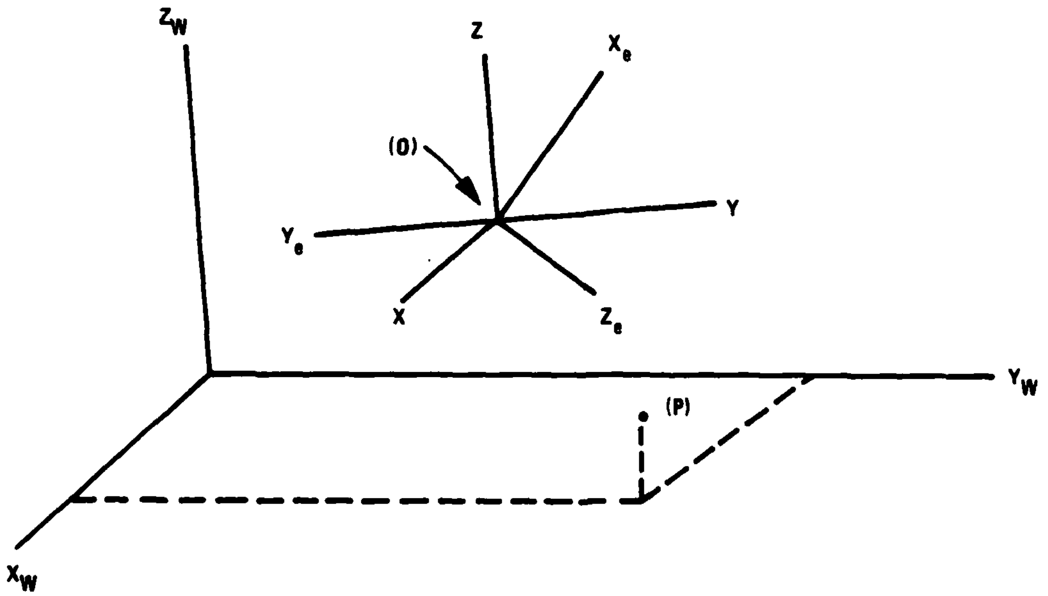


Figure 11. Image mapping coordinate systems.

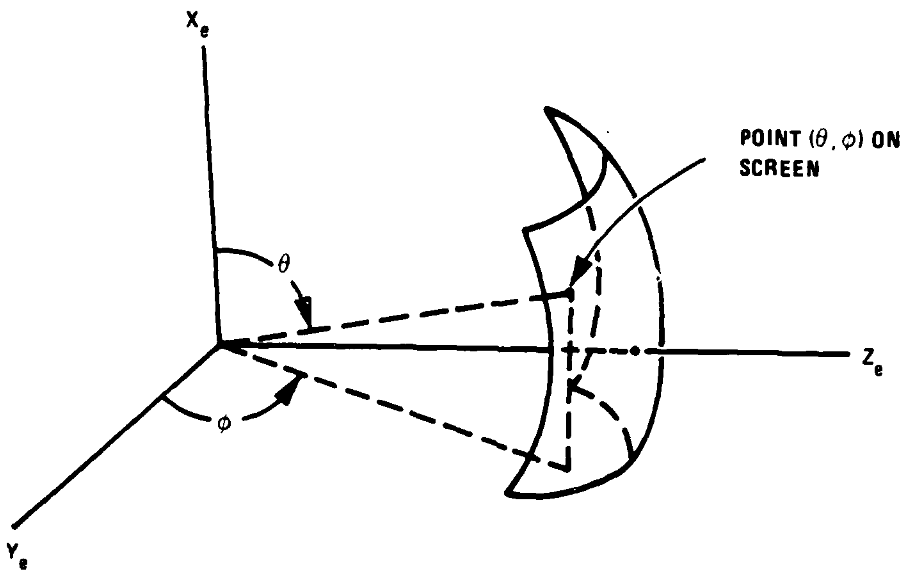


Figure 12. Spherical screen coordinates.

Finally, the screen coordinates (Θ,φ) are given by:

$$\Theta = \tan^{-1}\frac{(Y_e^2 + Z_e^2)^{1/2}}{X_e} \qquad \phi = \tan^{-1}\frac{Z_e}{Y_e}$$

Screen depth can be approximated by Z_e; no radius is necessary.

To avoid excessive computation time, both the arc-tangent and the square root should be determined by table look-up and linear interpolation. Unfortunately, the square root must be taken over a wide range, making linear interpolation inaccurate. If the square-root term is slightly inaccurate, then the arc-tangent will only exacerbate the problem. Since the (Θ,φ) coordinates must be quite accurately determined (within 1/5 pixel x 1024 pixels), the flat screen perspective transformation should be used instead.

3.1.8 Occultation

In the AVSS system proposed here, the resolution of priority conflicts is done by using a depth or distance buffer, or a z-buffer. This is done so features nearer the viewpoint properly occlude features farther away. A frame buffer holds the intensities of display points as they are generated. The depth buffer stores the distances of these same points and resolves priority conflicts by keeping both the intensity and distance of only the nearest point when conflicts occur. A conflict occurs when two terrain patches or objects project to the same position in the screen space. See Appendix A for a discussion of other occultation techniques.

The depth or distance used in the conflict resolution will be z_e rather the z_s. Briefly, z_e is the distance of a point from the observer's eye and z_s is a point's depth in the screen coordinate system. Using z_e for comparison, the values can occur in a broad range. Using z_s, the values are normalized between zero and one.

This approach is somewhat high-speed memory intensive. Items that must be considered here involve the amount of memory required and the memory layout.

The memory requirements are two frame buffers for intensity plus one frame buffer for the z-buffer. The calculation for this is:

1. Frame Buffer Memory
   2 * 3 colors 8 bits/color * 1024 * 1024 * 3 (anti-alias) $\cong$ 19 x 10^6 bytes

2. z-buffer
   1024 x 1024 * 24 bits (range) * 3 (anti-alias) $\cong$ 10 x 10^6 bytes

The total is 29 Mbytes of high-speed memory.

At this point, the system might time out unless some novel techniques can be used to get the data. For instance, on a low flight simulation, a large number of z-buffer tests will be made. More areas can be occluded as one gets closer to the ground with this simulator. The number of perspective transforms remains roughly constant.

For readout, the intensity memory can be a well-established bit plane structure using dynamic Random Access Memories (RAMs). During the intensity calculation, a buffer might be needed between the intensity calculation and frame buffer. For example, the screen space could be divided into vertical segments, each 16 or 32 pixels wide. Then the buffer mentioned would be approximately 300 kilobytes (Kbytes) of high-speed static RAM (see Figure 13).

A benefit of the z-buffer approach is that when the frame is displayed, the distance to each pixel is available for intensity modification due to haze, fog, or other atmospheric attenuation. Furthermore, no sorting is required (as in scan line algorithms), no ambiguities occur, and the constraint on resolution is the z-buffer size rather than the quickness of a sort in z.

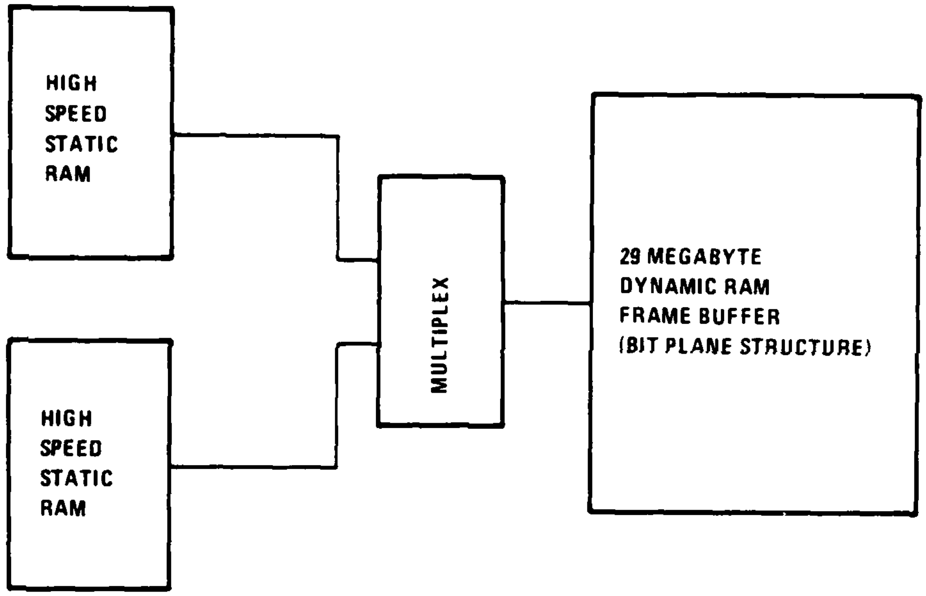


Figure 13. Frame buffer implementation.

3.2 IMAGERY CHARACTERISTICS

The most important area to be covered in the AVSS approach is imagery characteristics. The system developed as part of this study has the capability to produce imagery of much higher quality than that available with conventional edge-based systems. The general requirements are in regard to

1. Tonal computation
2. Texture
3. Terrain and hydrographic features
4. Cultural/features
5. Shadow
6. Special capabilities/considerations

3.2.1 Tonal Computation

In the AVSS concept, it is desired to generate imagery in the visual sense but also imagery from IR and LLLTV from the same data base. Each of these systems is considered in the following order: FLIR, LLLTV, and visual. The primary difference among the three systems is the tonal difference in black-to-white intensity or color due to the specific sensor characteristics. Hence the major problem is in the data base preparation and not in the perspective subdivision or real-time area. However, the intensity calculation will be affected by attenuation factors.

3.2.1.1 FLIR--The model for IR imagery requires that the temperature and emissivity be known for each object and material to be displayed. Given this information, the FLIR sensor output can be calculated. The total energy received at the detectors is the sum of all the contributions from the scene within the element's projection on the ground. The intensity is computed by dividing the radiant energy by pi.

The next step in the calculation is to compute the irradiance at the sensor aperture. Irradiance, H, is obtained by dividing the radiant intensity, J, by the slant range squared and multiplying by an atmospheric attenuation factor:

$$H = (e^{-\alpha R})\, J/R^2$$

The atmospheric attenuation factor, $e^{-\alpha R}$, models the effect of the atmosphere on the irradiance. The attenuation coefficient, α, is a function of the environment and the sensor waveband.

Finally, the calculation is completed by multiplying the irradiance by the
area of the sensor aperture ($\pi D_o^2/4$) and the optical efficiency:

$$P = \tau_o \frac{\pi D_o^{2}}{4} H$$

Note that this analysis assumes that the temperature is known. It depends on the time of day, sun radiance, cloud cover, moisture content in the background, and other factors. In order to generate the images in infrared, a model will have to be chosen and the effects of that model will appear in the simulation. Also, a lot of information must be added to the data base.

3.2.1.2 LLLTV--The low light level systems in use today are the silicon vidicon and its derivatives and fiber-optically coupled single-diode, double-diode, or micro-channel image intensifier systems. The simplest device is the silicon vidicon, which has a response extending into the near-infrared, especially when used with tungsten lighting. An object is seen against its background because there is a reflectivity difference causing contrast. The wavelength dependence of the contrast is determined by the materials involved. However, no one material can be characterized by a single reflectivity curve, so materials are usually characterized by their average reflectivity.

Several parameters must be considered in the generation of the LLLTV imagery. A few of these are the intensity of the illumination falling on the scene, the intensity of the illumination due to scattering, the atmospheric attenuation, and the reflectivity.

As the light level increases, the contrast in LLLTV decreases and it becomes increasingly difficult to pick out the areas of interest for cues.

3.2.1.5 Achromatic Intensities in the Visible Spectrum--Each flat surface element in an FOV requires an intensity value. This applies both to cultural flat surfaces representing houses, factories, etc and to the small flat quadrilaterals used to approximate curved terrain. This subsection will describe the method of applying intensities to computer-generated surfaces.

The most general real-time illumination model includes both specular and diffuse terms for the light reflected off a surface. The diffuse reflected intensity is given by:

$$I_D = I_A + R_D \max\left(0, \vec{N} \cdot \vec{L}\right)$$

where

I_D = diffuse intensity

I_A = ambient intensity

R_D = diffuse reflectivity

$\vec{N}$ = surface normal (normalized)

$\vec{L}$ = sun/moon direction vector (normalized)

The dot product must be thresholded at zero because of non-physical negative $\vec{N} \cdot \vec{L}$ values when the surface is facing away from the sun. Diffuse reflectivity is a function of the surface material and is a constant provided off-line. Ambient intensity is the intensity a surface will have in shadow. Thus it is a function of the weather, illumination source, and surface material, and is a constant provided off-line.

For the vast majority of intensity calculations (including terrain surfaces), the diffuse term alone is adequate for purposes of realism. For a few isolated objects, however, specular reflections are quite important. The specular component of the reflected intensity from a surface is given by:

$$I_S = R_S \max\left(0, (\vec{N} \cdot \vec{H})^c\right); \quad \vec{H} = \frac{\vec{L} + \vec{E}}{|\vec{L} + \vec{E}|}$$

where

I_S = specular intensity

R_S = specular reflectivity

c = shininess coefficient

$\vec{L}$ = sun/moon direction vector

$\vec{E}$ = viewing direction vector

Specular reflectivity and the shininess coefficient are surface-dependent, assigned off-line. $\vec{E}$ is the vector directed from the surface element to the viewer's eye, which must be recomputed each frame time.

The final factor in the intensity computation is the atmospheric attenuation with distance. The attenuation coefficient is wavelength-dependent and can be adjusted for other spectral regions. The final intensity equation for an achromatic surface is:

$$I_D = (I_D + I_S)\, e^{-\alpha r} + I_B\, (1 - e^{-\alpha r})$$

where

I_D = "observed" intensity of surface element

I_B = background (sky) intensity

α = attenuation coefficient

r = slant range (distance from observer to surface element)

3.2.1.4 Color--There is an abundance of theories relating human color perception and methods for producing color images. An HSV color space method is used here, for two reasons:

1. The data base must be enhanced to make accurate color pictures. This means associating some sort of color information off-line with terrain and cultural object surfaces. The HSV space is consistent with intuition and therefore tends to minimize errors and procedural confusion. Hue is the dimension normally described by adjectives like red, green, purple, etc; that is, it is what the layman calls color. Saturation is a measure of the departure of a hue from achromatic, that is, from white or gray. For example, red is more saturated than pink. Value measures departure from black, and for purposes of this report is equivalent to intensity as described above.

2. The simulation of shadow and atmospheric effects must be included. Both of these can be handled by the general equations:

$$H_0 = H \cdot e^{-\alpha r} + H_B(1 - e^{-\alpha r})$$

$$V_0 = V_L \cdot Se^{-\alpha r} + V_B(1 - e^{-\alpha r})$$

$$S_0 = S \cdot e^{-\alpha r} + S_B(1 - e^{-\alpha r})$$

where

H,S = hue and saturation of object surface on a (0,1) scale; assigned off-line

$V_{L,S}$ = value (intensity) of surface in light or shadow (L,S) computed each frame time

H_B,V_B,S_B = hue, value, and saturation of sky (depends on weather; assigned off-line)

The above formulation allows a simple weighted average among all quantities. This should be compared to the obscure problem of dealing with haze, say, in RGB space. Note that $e^{-\alpha r}$ is to be computed once per polygon surface element.

Z_e can be used in place of "r" as a time-saving approximation and will not be too inaccurate because the FOV is 30° on either side of the Z_e axis, and $Z_e \simeq r$ for large Z_e, where atmospheric effects predominate. Finally, $e^{-\alpha Z_e}$ can be determined by table look-up.

To summarize:

Information stored off-line; accessed for each surface element each frame time	H,S,I_A,R_D,R_S: Surface element hue, saturation, ambient intensity, diffuse and specular reflectivities. $H_B,S_B,V_B1\alpha,L$: Sky hue, saturation, value, attenuation coefficient, and sun/moon direction vector.
Information computed each frame time for each surface element	$N,E,Z_e \simeq r$: Surface normal vector, eye vector, and distance from eye to surface approximated by Z_e.

The above values are used to determine the final hue, saturation, and value of each surface element; H_O, S_O, V_O; where:

$$H_O = H \cdot e^{-\alpha Z_e} + H_B(1 - e^{-\alpha Z_e})$$

$$S_O = S \cdot e^{-\alpha Z_e} + S_B(1 - e^{-\alpha Z_e})$$

$$V_O = V_B\,(1 - e^{-\alpha Z_e})$$

$$+ \left[I_A + R_D \max\left(0, \frac{\vec{N} \cdot \vec{L}}{|\vec{N}|}\right) + R_s \max\left(0, \vec{N} \cdot \frac{(\vec{L} + \vec{E})}{|\vec{L} + \vec{E}|}\right) \right] e^{-\alpha Z_e}$$

For display with a device using the usual RGB color cube, H_O, S_O, and V_O need to be converted to RGB space quantities. The algorithm for this was published by Smith (1978):

$$H' = 6 \cdot H_O$$

$$I = \text{floor}\,(H')$$

$$F = H' - I$$

$$M = V_O\,(I - S_O)$$

$$N = V_O(1 - (S_O \cdot F))$$

$$K = V_O\,(1 - [S_O \cdot (1 - F)])$$

Then switch on I into

Case 0: $(R,G,B) = (V_O,K,M)$

Case 1: $(R,G,B) = (N,V_O,M)$

Case 2: $(R,G,B) = (M,V_O,K)$

Case 3: $(R,G,B) = (M,N,V_O)$

Case 4: $(R,G,B) = (K,M,V_O)$

Case 5: $(R,G,B) = (V_O,M,N)$

where

1. Floor (H') is the integer just less than or equal to H'.
2. Only one case is executed on the switch statement.
3. (H_O, S_O, V_O) and (R, G, B) are normalized on the range 0 to 1.

To do these color operations on a computer would require 13 multiplies, 13 adds, one divide, one square root, an integer operation, and an exponential multiply. For real time, a parallel pipeline organization will be used. At present, a block diagram is shown in Figure 9; the only item not shown is the Switch on I logic. The number of channels of this architecture has not been determined. The intensity calculation will have to be done on approximately 5×10^6 polygons for the mission area to be considered here.

3.2.1.5 Other Possibilities--The dot product for the intensity calculation must be normalized; that is, in $\vec{N} \cdot \vec{L}$, both $\vec{N}$ and $\vec{L}$ must be unit vectors. Since L can be made a unit vector off-line, the diffuse intensity can be written as:

$$I_D = I_A + \max\left(0, \frac{\vec{N} \cdot \vec{L}}{|\vec{N}|}\right), \text{ where } |\vec{N}| = (Nx^2 + Ny^2 + Nz^2)^{1/2}$$

$\vec{N}$ is determined for each surface element each frame time and adds negligible expense. Because finding $|\vec{N}|$ is computationally expensive, and there is no known simple and accurate approximation of it, this should be investigated in more depth before the above intensity model is adopted.

A further possible intensity model is one in which only certain special sun positions are allowed. For example, with a sun at 45° altitude in the northwest, Horn (1979) determined the intensity of a surface approximately with the following simple linear formula:

$$I(p,q) = 0.4285(p - q) - 0.0844\ |p + q| = 0.6599$$

where

p = slope in East-West direction

q = slope in North-South direction

Since p and q may be easily obtained in a recursive process of subtracts and adds, this model would be very quick. Future efforts should pursue a more flexible process. Perhaps similar linear formulas can be manufactured for arbitrary sun locations.

3.2.2 Texture

The simulation of terrain and object textures is an especially important task with respect to training effectiveness. It is clear, for instance, that a single tone surface representation alone gives the pilot no indication of distance. Furthermore, the movement of the edges of such regions over the pilot's FOV appears to provide inadequate depth information, judging from the failure of current systems to accurately simulate low-level, high-speed flight.

Although Gibson (1950) qualitatively demonstrated the importance of movement parallax in the ability of trained pilots to land aircraft, very few quantitative studies have been made. Of these, perhaps the most revealing is a theoretical analysis of the mathematical and visual invariants in movement parallax by Koenderink and van Doorn (1976). In the conclusion of their analysis they state that time change of texture density conveys information about orientation and distance.

Further support for the contention that texture is important as a distance cue, especially for nearby objects in low-level flight, is found in Bajcsy, Friedman, and Sloan (1976). In their abstract they state that texture gradient gives a qualitative value for distance.

Thus it appears that in a moving environment, realistic changes in texture provide distance and orientation information. This conclusion has served as the primary motivator in the design of an appropriate texture generation algorithm.

5.2.2.1 Terrain Texture--Our primary texture generation methodology is an extension of the fractal techniques introduced by Mandelbrot (1977) and developed by Fournier and Fussell (1980) and Carpenter (1980). The methods of these authors represent natural irregular objects and terrain by modeling them as sample paths of stochastic processes. The particular stochastic process is called "fractional Brownian motion." The implementation of fractal texturing allows computing the surface to arbitrary levels of detail without increasing the data base. Therefore, the simulation will result in a continuous change of detail and texture density for all flight paths.

In addition, Fournier's technique ensures that macroscopic texture features will remain identical while finer details in the texture change, preserving object or texture identity. This occurs because his method requires seeds for random number generators. These seeds are linked to patch corners to assure coherence across patch boundaries within a single frame, and the seeds do not change from frame to frame. This means that surface perturbations at patch corners are fixed. Interior perturbations are computed by recursive subdivision, using the result of the previous subdivision as the seed for the next point. Thus, if the subdivision level for a patch is not changed from one frame to the next, the surface perturbations are identical. But if the level of detail does change, the larger perturbations are maintained and only the finest perturbations are altered.

Off-Line Generation of Texture Labels--Preparatory to real-time texture generation with fractals is an off-line procedure which begins with a list of DMA perimeters of areas designated by different surface material categories (SMCs). The procedure ends with each 100-meter-square area (loosely called a patch) in the mission area being assigned a texture code. Part of the off-line generation is derived from traditional chain coding techniques, Rosenfeld and Kak (1976), and the rest is a Honeywell addition. A broad outline of the off-line texture label procedure is diagrammed in Figure 14; a more detailed narrative description follows:

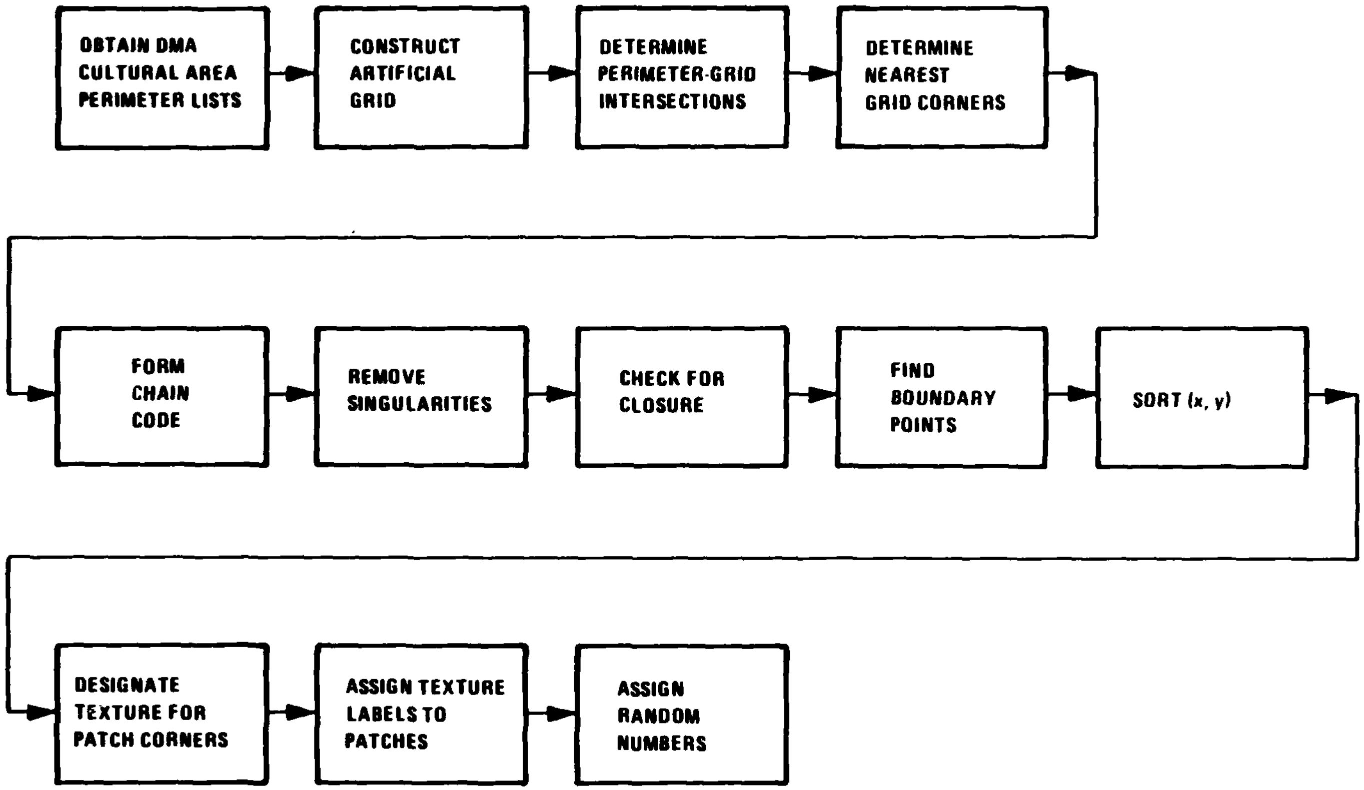


Figure 14. Off-line generation of texture labels.

1. The cultural part of the DMA data base provides lists of area perimeters, where each list contains clockwise-ordered (x,y) coordinates of several points serving to outline the area. The coordinates are in latitude and longitude offsets from a reference point. The DMA notes that the border definition is not guaranteed to close if the region border extends beyond the manuscript edge. Testing for closure and correction takes place immediately after the construction of a chain code border representation.
2. Chain Code Construction: Figure 15 illustrates an example of a perimeter defined by a small number of points. To chain code this perimeter, an artificial grid is constructed on the same scale as the DMA elevation data base, so that each corner in the artificial grid falls in the center of each DMA terrain patch.

Starting with the first (x,y) point in the list, traverse the perimeter clockwise and at each intersection of the perimeter with a grid line, determine the nearest grid corner. In Figure 15, the first such corner is at A, the second is at B, etc. The chain code itself is an eight-directional code which specifies the directions between successive grid corners. For example, in going from A to B the code is 0, from C to D it is 1; and from H to I it is 4.

Redundant information from successive intersections giving the same corner is excluded. For instance, near the corner I there are two intersections; the second intersection is ignored.

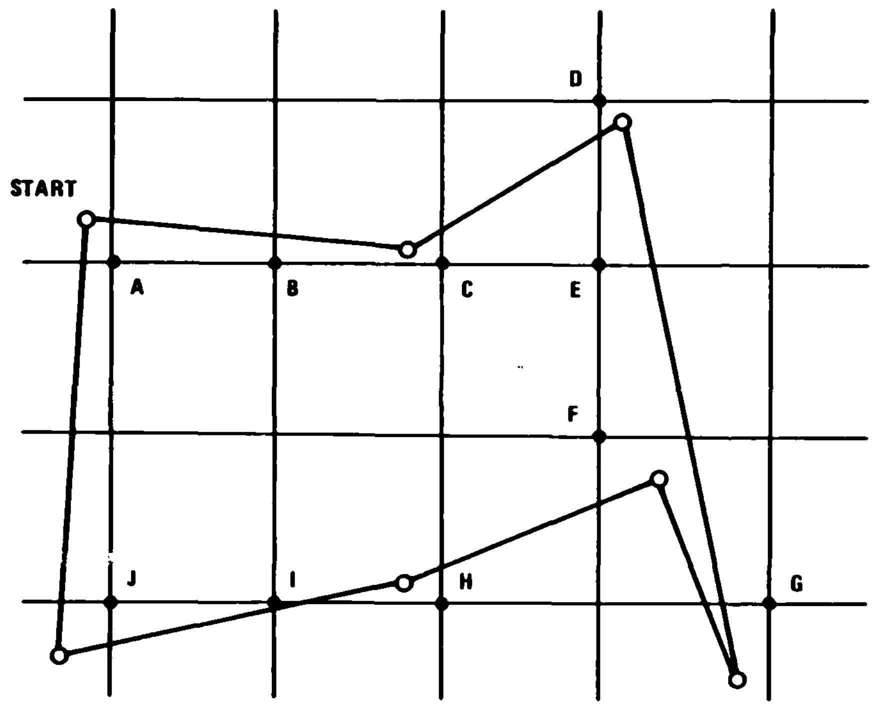
Finally, singularities in the code must be removed. In our example, a singularity exists where the nearest corners are successively F-G-F, leading to a chain code of 73, a line segment that reverses direction. Any time opposite chain code number pairs (40, 51, 62, 73) occur, they will be excluded.

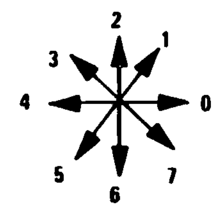
Once the final chain code is constructed, it can be tested for closure by the following rule, which applies to chains of arbitrary length:

> Rule for Closure: If $n_1 + n_2 + n_3 = n_5 + n_6 + n_7$ and $n_3 + n_4 + n_5 = n_7 + n_0 + n_1$ are both true, then the chain (and boundary) is closed; where n_i = number of chain code numbers (i).

(Note that throwing out opposite chain code number pairs leaves the above rule intact.)

Intersection Procedure--The determination of grid line and boundary intersections used for chain code construction is accomplished a line segment at a time. Each line segment is specified by its two endpoints. Knowledge of these endpoints allows the specification of all horizontal and vertical grid lines which will be





CODE WITH SINGULARITY	CODE WITHOUT SINGULARITY
0	0
0	0
1	1
6	6
6	6
7	5
3	4
5	4
4	2
4	2
2	
2	

Figure 15. Chain code for a standard texture border.

intersected by the line segment. Explicitly, given the endpoints P_1, P_2 = (x_1,y_1), (x_2,y_2), the vertical grid lines intersected by the segment $\overline{P_1P_2}$ will be:

$$X = K1 + K2(i_n)$$

$$X = K1 + K2(i_{n+1})$$

$$X = K1 + K2(i_{n+2})$$

etc

where K1 and K2 are constants and i_n is some integer value such that X falls between X_1 and X_2.

Similarly, the horizontal grid lines intersected by $\overline{P_1P_2}$ are given by:

$$y = K3 + K4(i_m)$$

$$y = K3 + K4(i_{m+1})$$

etc

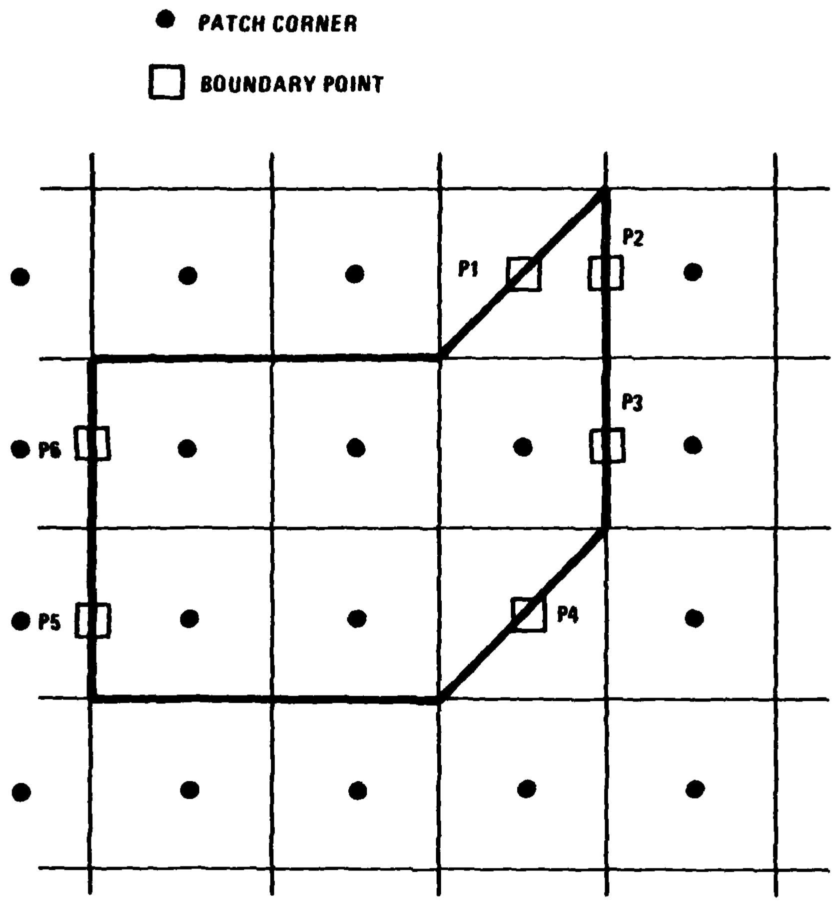
where y falls between y_1 and y_2.

It is essential that the intersections fall in an ordered sequence progressing from P_1 to P_2, so that the chain code makes sense. Because the intersections all lie along a line, they can be sorted according to ascending x values when $x_2 > x_1$ and descending x when $x_2 < x_1$. Also, because the number of intersections is usually quite small (<50) in terms of sorting, and the x and y lists are closely interleaved, a merge-sort which is approximately (n log n) in time will suffice. Once the ordered intersections are obtained, nearest grid corners can be computed by simple compare operations.

Boundary Points and Patch Corner Designation--Once the chain code for a perimeter is established, it can be used to determine what are called "boundary points" and these can be used to determine texture labels. The procedure can be understood by studying the example in Figure 16. The heavy line represents the perimeter drawn from the chain code in the previous figure.

Patch corners are represented by x's in Figure 16 and they fall in grid centers. What must be known is just which patch corners are inside the perimeter. For convenience, patch corners which fall on diagonal perimeter line segments are counted as interior points.

The procedure is to first obtain boundary points P1, P2, P3, P4, P5, and P6, which occur on vertical or diagonal boundary segment midpoints. This can be easily accomplished with a starting (x,y) coordinate and the chain code. These points are then sorted twice--first in y, then in x--to obtain the lists shown in Figure 16. The final list is grouped in pairs (P_1,P_2), (P_6,P_3), and (P_5,P_4). All patch corners with x values falling between these



INITIAL BOUNDARY POINT LIST	...AFTER Y SORT	...AFTER X SORT
P1	P1	P1
P2	P2	P2
P3	P3	P6
P4	P6	P3
P5	P4	P5
P6	P5	P4

Figure 16. Boundary points and texture labels.

pairs are counted as being inside the perimeter. The result is a list of patch corners which have the texture designation of the perimeter's SMC code.

It should be noted that this method always has boundary point pairs and not single boundary points because patch corners falling on diagonals are counted and because singularities are eliminated during the chain code construction.

Texture Labels--Once all patch corners in the mission data base are assigned texture labels, patch corners can be taken four at a time to indicate whether a patch is all one texture or partly two. If all four vertexes have the same label, the corner labels are discarded and the patch receives just one label designating its texture. During the mission simulation, the label is used as a pointer into a fractal lookup table, allowing it to be textured.

If one, two, or three vertexes have one texture label and the others have a different label (or none), the corner labels are saved for each patch, and the patch is given the special title "texture border patch."

One final off-line procedure is necessary. Every patch corner with a texture label is assigned a random integer x, where $-5 \leq x \leq 5$. It is essential that this be done to the patch corners before they become part of a patch tree structure, so that four patches sharing one corner also share the same random integer assigned to that corner, and so that fractaled terrain appears continuous.

3.2.2.2 Real-Time Terrain Texture Generation--During the mission run, patches that have a single texture label will be textured by the fractal subdivision method described below. Each texture will have a pointer into a specific fractal lookup table. Thus, for instance, water and marsh will be represented by two different lookup tables. The number of lookup tables will equal the number of textures, except for special cases like snow under certain weather conditions, which will simply be assigned a high reflectivity. A particular advantage of this method is therefore the low storage required.

Fractals--The usage of the term "fractals" is much broader than the original meaning. Mandelbrot (1977) intended fractals to stand for self-similar statistical functions which were nowhere differentiable. In a strict sense, the definition used here refers to partly differentiable statistical functions which are not necessarily self-similar. However, a less prosaic and more useful definition is a computer-generated surface or curve which employs linear recursive subdivision and an LUT.

A typical fractal LUT appears in Figure 17. It can be used to generate either fractal curves or surfaces. The procedure for generating a fractal surface consists of generating several fractal

	-5,5	-4	-3	-2	-1	0	1	2	3	4
-5,5	-3	0	1	2	-1	3	0	-1	2	-3
-4		-2	-1	0	2	1	-2	4	-1	-1
-3			-3	-1	-2	-3	-1	-0	-2	-0
-2				-1	-1	-1	-0	-4	-2	-3
-1					-0	-4	-1	-1	-0	-1
0						-2	-0	-4	-5	-1
1							-5	-0	-1	-2
2								-1	-2	-1
3									-2	-0
4										-1

Figure 17. Fractal look-up table.

curves in parallel and combining them. The procedure for generating a fractal curve along a patch edge is outlined below. Each step >3 in the outline will have a numerical example which follows the LUT in Figure 17 (steps 1 and 2 have arbitrary numbers).

1. Start with the two endpoint elevations of the patch edge: (0,0)

2. Add to these corner elevations the random numbers assigned to the corners off-line: (-1,2)

3. Using the new elevations, find the midpoint value:

$$\frac{0 + (-1) + 0 + 2}{2} = 1/2$$

4. From the LUT read out the matrix value corresponding to the row-column pair: (-1,2) = 1 (Note that (-1,2) act as seeds)

5. Add this value to the midpoint to create a new midpoint elevation: 1/2 + 1 = 1-1/2

6. If only two segments are needed for display, then their elevations are given at their endpoints: (-1, 1-1/2, 2) = E_1, E_M, E_2

7. Seeds used to generate these elevations are always stored with the elevations:

$$[(E_1,S_1),\ (E_M,S_M),\ [(E_2,S_2)]] = [(-1,-1),\ (1\text{-}1/2,1),\ (2,2)]$$

8. Subdividing the left section requires the seeds (-1, 1) for which the correction factor is one. Steps 3-7 are continued recursively down to any level desired. At each step, the correction factor can be divided by some integer power of two, so that at a high level of detail extremely high slopes are avoided. In general, when the integer power equals the level of subdivision the fractal curve will be self-similar. However, other rules can be used to generate arbitrary textures, including substituting absolute values of correction factors for the correction factors at given subdivision levels. Also, different kinds of LUTs can be constructed (the one used in this study consisted of numbers ranging from -5 to 5, with a Gaussian distribution about a mean of zero; a Gaussian distribution may not be necessary for many textures).

Fractal surfaces are easily formed by constructing fractal curves for each patch edge and diagonal. For instance, to subdivide a patch once, the procedure would be: Given corners A,B,C,D and in clockwise order the edges AB, BC, CD, DA; find midpoints and correction terms for each edge, and find the midpoint and correction term for the diagonal AC. These five midpoints plus the four original corner points designate four new subpatches.

Real-Time Terrain Texture Border Generation--Recall that some patches will be labeled "texture border patches" off-line because some of the patch corners have one texture label and the other corners have another label. These patches will be treated with a fractal subdivision method analogous to that just presented, but in a more abstract way. The purpose will be to construct a ragged-texture border about an area of homogeneous texture. This aids in realism, is relatively inexpensive, and perhaps most significantly, reduces the number of straight edges (hence aliasing) in the simulation.

The idea central to the construction of a rough-texture border is to create a series of interconnecting patch border segments. Thus, each patch border segment must meet two requirements:

1. The border segment is rough and statistically similar to all other border segments.
2. The border segments are continuous from patch to patch, giving the appearance of one continuous, closed border.

Both these requirements can be met by the application of fractal subdivision to texture border patches. As a heuristic aid, consider the border patch in the top of Figure 18. Each corner has a particular texture label and a particular random integer x, where $-5 \leq x \leq 5$. These labels and numbers will serve to generate three fractal

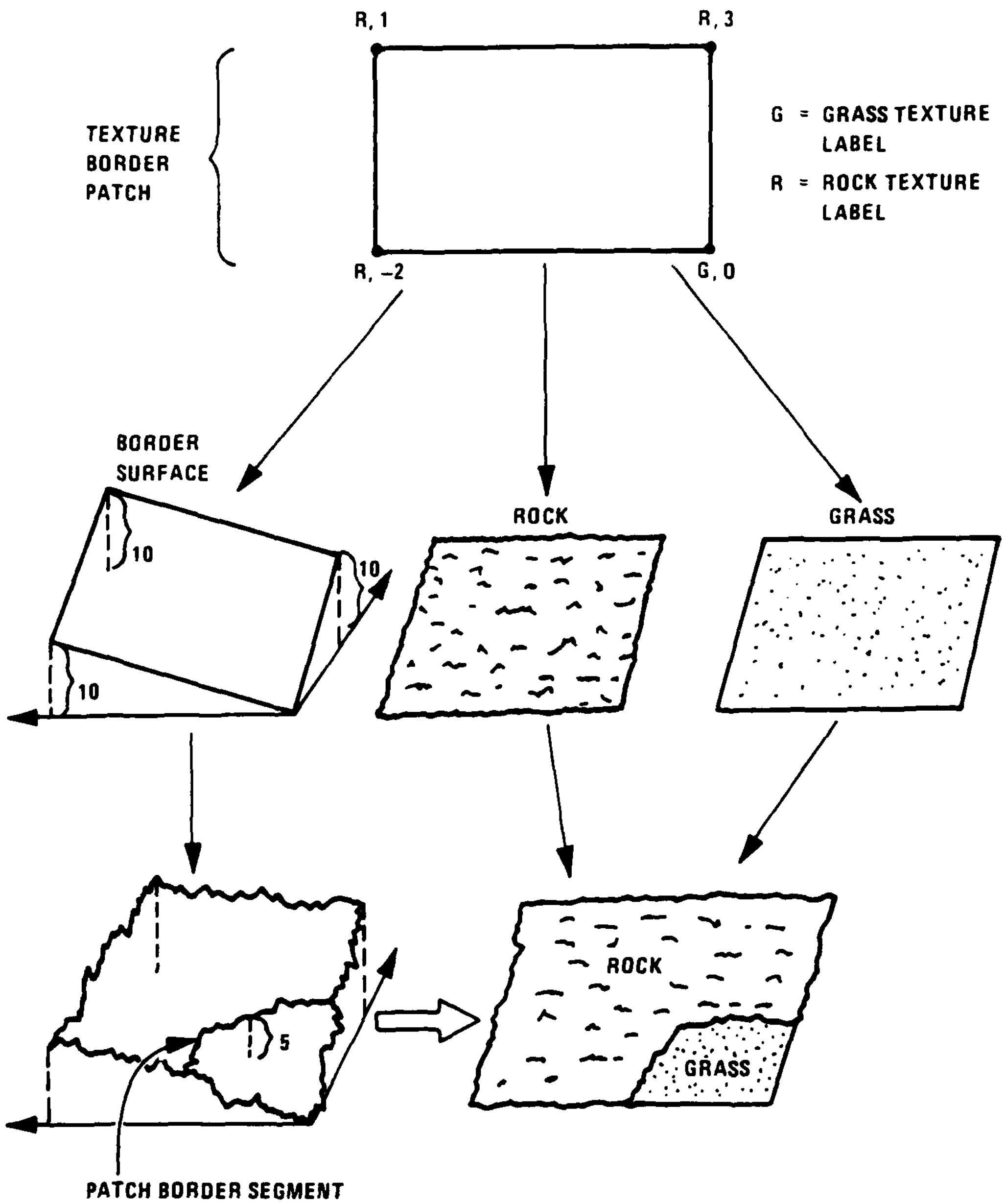


Figure 18. Border patch subdivision.

surfaces in parallel corresponding to the one patch. One surface will be the grass surface and the second a rock surface, both created though not necessarily displayed over the entire patch. The third fractal surface is not displayed, it is constructed with so-called "false elevations" and is used only to delimit the border between grass and rock surfaces. This is accomplished by assigning a zero false elevation to one texture label and an elevation of 10 to the other (it does not matter which is which as long as it is consistent). Adding fractal perturbations to this tilted surface during fractal subdivision will sometimes result in subpatches with corners having elevations >5 and at other times <5. In the former case (see Figure 18), the subpatch corner is designated with an R = rock code and in the latter case, a G = grass code.

It is important to note that the level of subdivision of all three patches is always the same, but that each patch has its own fractal look-up table, so that the initialized random integers shared by the three patches produce three different textures.

Two real elevations are computed and available for each subpatch corner. Depending on the false elevation value, one of the real elevations corresponding to one of the textures is chosen. When these elevations are eventually used for display, subpatches containing corner elevations from both textures will have intensities assigned according to an average of texture reflectivities and hues, or just one dominant reflectivity and hue; the final arbitrator will be visual tests of DMA texture border pairs.

It should be noted that even when a patch or subpatch has elevations from different textures' LUTs, surface continuity is preserved; that is, no gaps appear. Furthermore, the LUT used for border construction may lead to vastly different borders depending on the nature of the LUT. Some LUTs may lead to fairly smooth, curved borders, others to extremely ragged borders. Many will lead to splotchy borders, which consist of many small, isolated clumps of one texture that gradually evolve into clumps of the isolated adjacent texture, so that no distinct border exists. This last case is controlled by the choice of the initial false elevation, which need not be 10 (and is quite arbitrary), by the LUT structure, and by the division factor for each level's fractal correction term. Finally, and briefly, the very rare cases where three or four textures fall on one patch can be treated by adding additional false elevation patches computed in parallel, or by arbitrarily eliminating them during the off-line preparation.

Texture Border Smoothing--In some isolated cases realistic borders will be gradual rather than distinct; for example, marsh to water. These cases can be treated by a more sophisticated border algorithm, in which the values of false elevations are used as weights to apply more of one texture than another. Thus, adjacent textural hues, saturations, and reflectivities could merge gradually according to the raise surface elevations. This procedure would consume little time because of its isolated application and so it is excluded from an operations analysis.

Texture Mapping--Storing a file of an image and mapping it to a surface is a particularly easy way to texture both terrain and cultural object surfaces. For application with terrain patches which can be projected to screen space at arbitrary levels of subdivision, a tree structure is required to store the textural image at many levels.

There are two defects to this technique: 1) for large areas it requires tremendous storage, and 2) only a finite amount of detail can be stored. As a consequence, close surfaces will appear blocky--or if averaged or smoothed, unrealistic and blurry.

On the other hand, for large areas requiring large-scale texture only, texture mapping is recommended. The best example of this circumstance is where large rectangular agricultural areas are present. In such a case, a patch or even several patches will be assigned a certain hue, saturation, and reflectivity to simulate the crop grown there. These assignments must be made off-line on the basis of photographic/statistical information for the regions in question. No specific methodology is suggested here.

Texture mapping is best applied to certain objects. For example, a roof of a house can have a small texture map with few branches in its tree because the house is inevitably only a few pixels or tens of pixels across. Such maps can be accessed for many objects simultaneously; for example, a hundred houses in one FOV would have pointers to only one texture map. Thus storage is minimized. Finally, texture mapping to cultural surfaces should only be attempted if absolutely necessary for pilot recognition.

Video Texture--At the time of this writing, video texture techniques have been tested successfully in a non-real time mode at Honeywell, but it is not known whether such techniques can be applied in real time at a reasonable cost. Studies are currently under way to answer this question. Assuming that video texturing techniques are feasible, they have the following potential applications:

1. The distant sky can be represented by a rotated, translated photograph representing any sky condition, including night skies with real star positions.
2. Individual objects can be modeled by rotated, translated, and magnified photographs of the objects themselves. A single tree, for example, could be represented by two photographs: a side view and a top view. One view would gradually metamorphose into the other (fade in/fade out) as the pilot's orientation above the tree changed.

3. Surfaces of large extent can be synthesized. For example, an entire sky of clouds would be generated by a perspective view of one photograph. Similarly, terrain roads could be overlaid on the terrain surface by mapping a perspectively correct road image to the simulator screen.

3.2.2.3 Texture Off-Line Operations--

1. Determine Perimeter-Grid Intersections:
   a. Given segment endpoints, find slope, intercept: 3s*, 1d, 1m per segment x N; N = number of segments in a perimeter
   b. Determine horizontal and vertical grid lines which will be intersected: 2P compares, where P = length of perimeter in grid units
   c. Find perimeter-grid intersections: (2m, 2a) x P
   d. Sort intersections (merge): P compares
2. Determine nearest grid corners: 2P compares
3. Form chain code: 2P compares
4. Remove singularities: P compares
5. Check for closure: number of adds roughly = 2 number of entries in chain code list 2P adds.
6. Find boundary points (easiest way is to save coordinate pairs from step 2 and find midpoints according to chain code) (1a, 1 shift x 2 per boundary point x number of boundary points Pa, Ps
7. Sort (x, y); quicksort on two keys: n log n x 2 compares where n P
8. Designate texture for patch corners: 1 operation (add) per patch corner x number of patch corners = area of textured region x 1a
9. Assign texture labels to patches: 3 compares per patch x area of texture
10. Assign random numbers to patch corners: 1 assign per vertex x area of texture

With a textured area of perimeter P, area A, and number of perimeter segments N, the total number of operations is:

---

*s = subtract, m = multiply, a = add, d = divide

3N subtracts
N divides
N multiplies
8P + 2P log P compares
6P adds + P shifts
2P multiplies
P shifts
A adds
3A compares
A assignments

Equating computation times for compares, adds, assignments, shifts, and subtracts to 0.38 μsec, multiplies to 11.2 μsec, and divides to 14 μsec, the amount of time required is:

$$1.9A\ \mu sec + 26.34N\ \mu sec + 0.76\ \mu sec\ (8P + P \log P) + 22.4P\ \mu sec$$

A typical textured area might be several kilometers in extent, for which the following is chosen:

$A = 5\ km^2 = 500$ in 100-meter size units
$P = 12\ km = 120$ in 100-meter increments
$N = 15$ segments

The off-line time to compute textural information for one area is then:

$$500(1.9 \times 10^{-6}) + (26(15 \times 10^{-6}) + 0.76 \times 10^{-6}(960 + 120(2)) + 22.4 \times 10^{-6}(120) = 5 \times 10^{-3}\ sec$$

Thus for a mission data base containing even thousands of textures, the preparation time is brief.

3.2.2.4 <u>Real-Time Operations to Produce Textured Surface Patches</u>--Adding fractaled texture to a surface patch involves a recursive subdivision procedure. Each subdivision requires:

$$\left.\begin{array}{l}\text{5 table look-ups}\\ \text{5 shifts}\\ \text{10 adds}\end{array}\right\}\ \text{times approximately } 10^6 \text{ subdivisions per frame}$$

Fractal subdivision architecture is described in Figure 19. Alternative texturing techniques are discussed in Appendix B; the question of using image space vs object space is discussed in Appendix C.

3.2.3 <u>Terrain and Hydrographic Features: Surface Representation</u>

The purpose of the Honeywell-Catmull algorithm is to create a parametrically-defined, continuous, curved surface which closely follows the discrete terrain elevation samples in the DMA data base.

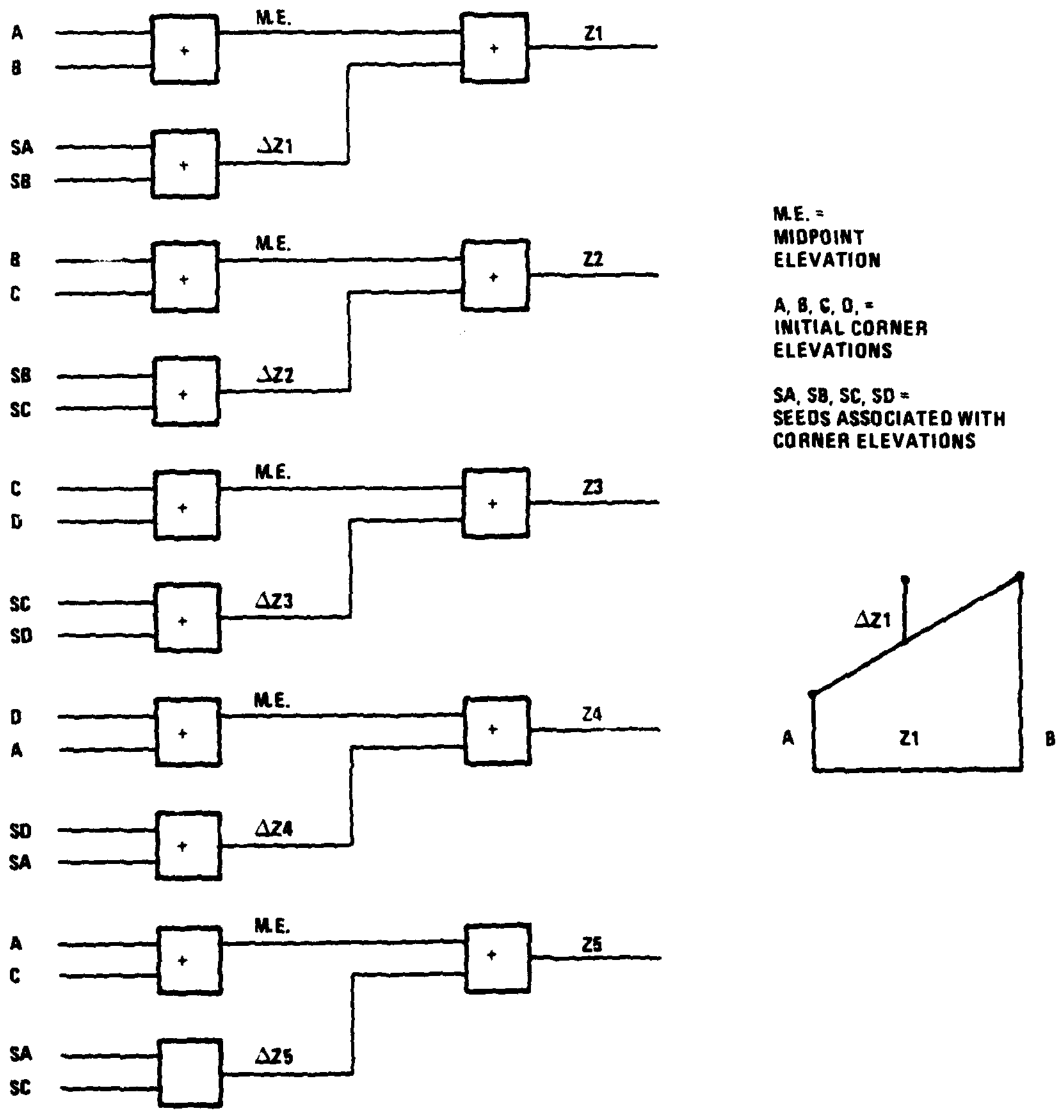


Figure 19. Fractal subdivision architecture.

The data base gives elevations at uniform grid intervals. Estimates of intervening elevations are needed for two reasons:

1. To compute the perspectively correct screen position
2. To compute the surface normal, which is used in the illumination model to compute intensity

The method for estimating the intervening elevations is essentially Catmull's (1974) method. He fits a bicubic spline surface to the sample elevations.

It is quite time-consuming to determine elevations from straightforward substitution into a bicubic equation, so Catmull devised a fast technique which recursively divides patches in half in latitude and longitude and finds the elevations by a clever combination of add, subtract, and shift operations. Remember that the reason for subdividing is to get an elevation estimate for a displayable piece of the terrain.

5.2.3.1 Method Outline--

I. Just as a flat surface can be constructed from many contiguous rectangles, define the entire curved terrain surface as a collection of many contiguous, curved "surface patches." (Technically this is called a spline surface.)

II. Prepare the surface patches so that they can be divided into four subpatches; as can each subpatch, recursively; down to any level necessary.

III. Each frame time, patch vertexes are perspectively transformed to the "screen space" of the pilot. (A "window" which corresponds to this screen must be constructed; see Figure 20).

IV. A rectangle is constructed to enclose the patch vertexes. The rectangle lies in a plane defined by the screen coordinates (x_s, y_s, $z_s = 0$); see Figures 21 and 22.

V. If the rectangle for any patch is pixel size or smaller, the pixel associated with the rectangle is given an intensity which depends upon the relative patch, pilot, and sun orientations, in addition to the texture and shadow assigned to the patch.

VI. If the rectangle is larger than pixel size, the corresponding patch (in world coordinates) is divided into four subpatches. Each subpatch vertex then undergoes a perspective transformation to screen space, with rectangle tests and subdivision occurring recursively until all rectangles are approximately pixel size. These tests occur each frame time.

VII. When the orientation of a patch or subpatch surface is facing away from the pilot, the patch or subpatch is discarded.

VIII. When the enclosing rectangles of two patches overlap, the area closer to the pilot is displayed.

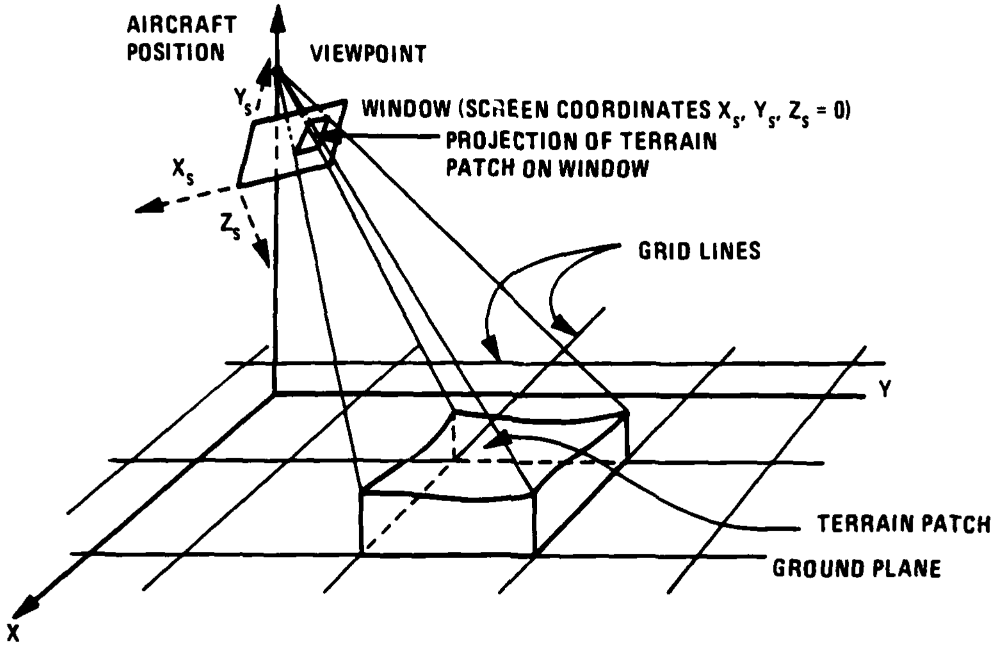


Figure 20. Terrain imaging geometry.

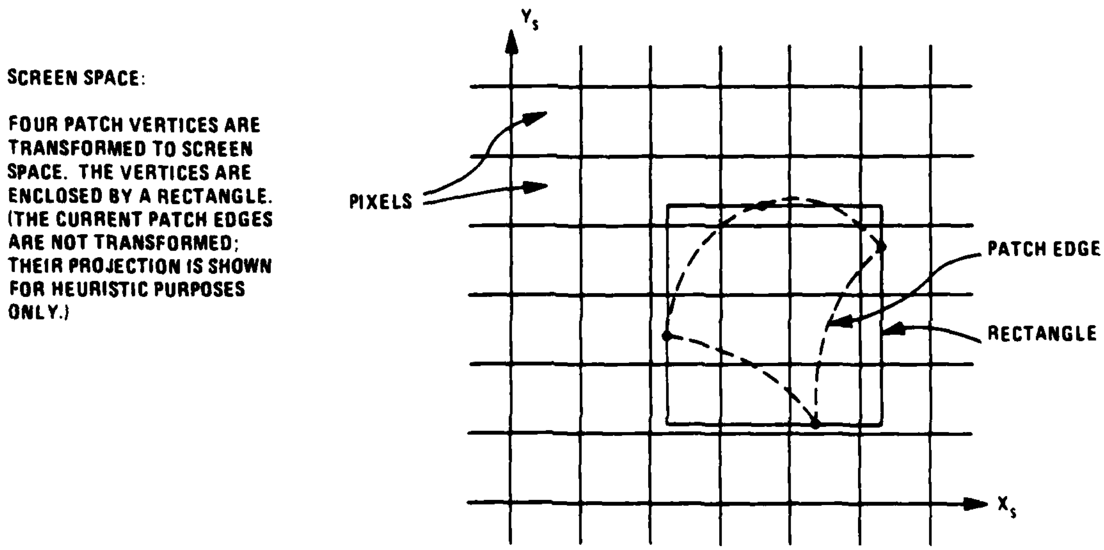


Figure 21. Patch size test.

IF THE RECTANGLE IS TOO LARGE, THE PATCH IS DIVIDED INTO FOUR SUBPATCHES AND THE VERTICES OF EACH OF THE SUBPATCHES ARE PROJECTED TO SCREEN SPACE. EACH SUBPATCH IS ENCLOSED BY A RECTANGLE. AGAIN, IF THE RECTANGLE IS TOO LARGE, THE SUBPATCH IS DIVIDED AND THE VERTICES PROJECTED. IF THE RECTANGLE IS APPROXIMATELY PIXEL SIZE, THE PIXELS ARE GIVEN APPROPRIATE INTENSITIES. DIVISION TO PIXEL SIZE ENSURES CONTINUITY OF INTENSITIES ACROSS THE SURFACE.

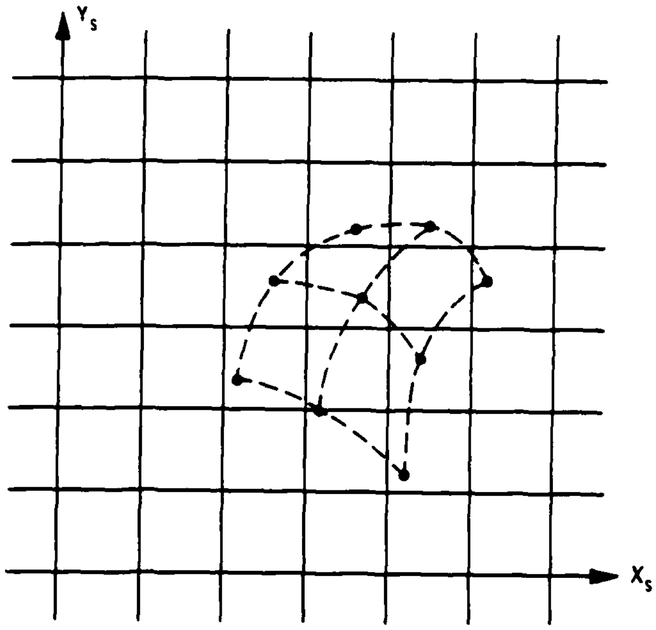


Figure 22. Patch too big.

The above description is intended only to provide a rough idea of the framework of the surface representation algorithm. A detailed description follows.

1. After the DMA elevations for the mission area are obtained, they must be accessed in groups of 16 adjacent elements and placed in 4 x 4 matrixes. Each group of 16 elevations is used to generate a curved bicubic surface patch whose borders are defined by the central four elevations among the 16. This is illustrated in Figure 23. Note, in generating an adjacent patch, that of the 16 elevations applied in its construction, 12 are redundant from the previous patch.

2. The mathematical equation of a bicubic, Elevation = $a_1u^3v^3 + a_2u^3v^2 + \ldots$, is entirely specified by its 16 coefficients $a_1, a_2, \ldots b_1, b_2, \ldots c_1, \ldots d_3, d_4$. Different kinds of bicubic surfaces can be manufactured; that is, different coefficients determined, depending on what we do with the elevations. To ensure elevation, first, and second derivative continuity across patch borders, a so-called B-spline matrix is applied to the elevation matrix. The specific operation is described in Figure 24.

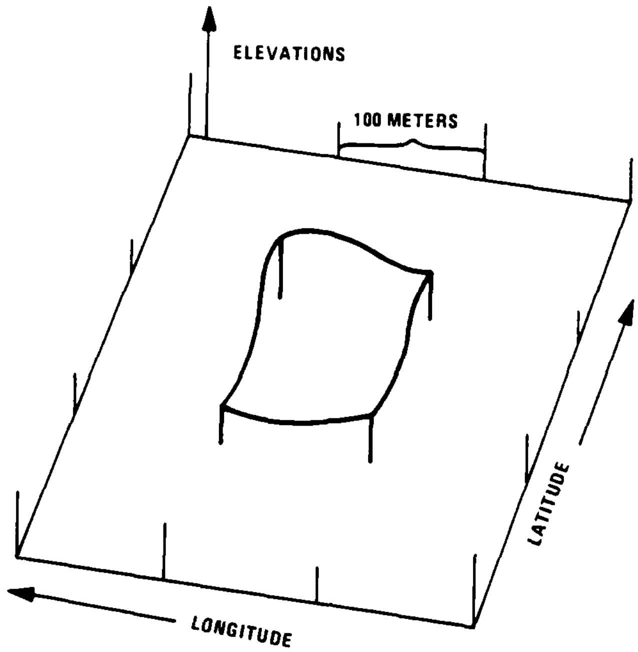


Figure 23. Bicubic patch and its determining elevations.

The matrix M in Figure 24 is for B-spline basis functions. It ensures that across all patch borders there is elevation and first and second derivative continuity. It is possible to construct patches with other matrixes. Some are interpolating, others are approximating. The B-spline approximates the control points but also provides more local control than any other approximating basis.

It is important to realize that the bicubic representation is the lowest-order polynomial representation of a free-form, curved, continuous surface. Lower-order polynomials do not ensure first or second derivative continuity at patch borders.

3. Succeeding the construction of a coefficient matrix, the coefficients are combined into "register squares." There are four register squares per patch, one for each corner. Each register square contains four numbers; for example, at the corner $u = 0$, $v = 1$ the four numbers are $(a_1 + a_2 + a_3 + a_4)$, $(b_1 + b_2 + b_3 + b_4)$, $(3a_1 + a_2)$ and $(3b_1 + b_2)$.

1. Place 16 elevations in a 4 x 4 matrix E. The 16 elevations come from the data base and therein are arranged in a 4 x 4 matrix.

2. Using a cubic B-spline matrix M, compute MEM^T where T means transpose.

$$M = \frac{1}{6} \times \begin{bmatrix} & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Then MEM^T forms a coefficient matrix:

$$MEM^T = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}$$

3. Compute register square values for each *patch corner.* A register square has four values, initially computed from the coefficient matrix, subsequently computed by subdivision.

u=0 v=1

$d_1 + d_2 + d_3 + d_4$	$b_1 + b_2 + b_3 + b_4$
$3d_1 + d_2$	$3b_1 + b_2$

u=1 v=1

(sum of all coefficients)	$3(a_1 + a_2 + a_3 + a_4)$ $+(b_1 + b_2 + b_3 + b_4)$
$3(a_1 + b_1 + c_1 + d_1)$ $+(a_2 + b_2 + c_2 + d_2)$	$3(3a_1 + b_1)$ $+ 3a_2 + b_2$

PATCH

u=0 v=0

d_4	b_4
d_2	b_2

u=1 v=0

$a_4 + b_4 + c_4 + d_4$	$3a_4 + b_4$
$a_2 + b_2 + c_2 + d_2$	$3a_2 + b_2$

Figure 24. Making register squares.

Register squares form the core of the Catmull subdivision algorithm. The idea is to start with only the patch corner elevations and some correction terms, then determine, by a recursive procedure (along the four patch edges and in its center), midpoints that lie on the bicubic surface. By continually determining midpoints for each subpatch, the collection of elevations obtained can serve to approximate the curved bicubic surface.

Each register square contains one corner elevation and three correction terms. Subdividing a patch into four subpatches implies turning four corner elevations into nine corner elevations and likewise four register squares into nine register squares. This process is diagrammed in Figure 25. The elevations required at each level of subdivision are always in the upper left corner of the register square.

5.2.3.2 Tree Structure--There are actually two distinct tree structures in the terrain display algorithm, one stored and one computed. The stored tree is the mission data base. It consists of the data necessary to display large surface areas, extending from the mission data base root node itself down to individual bicubic patch corners. The lowest nodes or terminal nodes in the stored tree are, therefore, bicubic patches.

The computational tree consists of the data necessary to display smaller surface areas, where the root nodes are the bicubic patch terminal nodes of the stored tree. Lower nodes in the computed tree are, as the designation implies, computed (if necessary) by recursively subdividing the root patch into small subpatches.

The computational tree is scanned in depth first-order; that is, the algorithm steps through the tree node to node, beginning at the top or root node. Nodes near the top represent large ground areas whose projections on the display screen will depend on orientation and distance from the pilot's viewpoint. Projections of nearby nodes will usually cover many pixels, forcing a scan to a greater depth for such nodes. Distant nodes will project to smaller picture areas, and the scan depth will be less. This feature allows an automatic selection of level of detail as well as minimizing the computational effort in producing a picture.

5.2.3.3 Picture Generation--The procedure for generating a picture is as follows. Starting with the top node, the picture area corresponding to the node is determined by projecting the four corners of the patch represented by the node to display screen coordinates. A decision is made to terminate the scan at that node or proceed to the next level. Termination can occur under the following conditions:

1. When the projected area represented by the node falls outside the display window

2. When the projected area is small enough to display

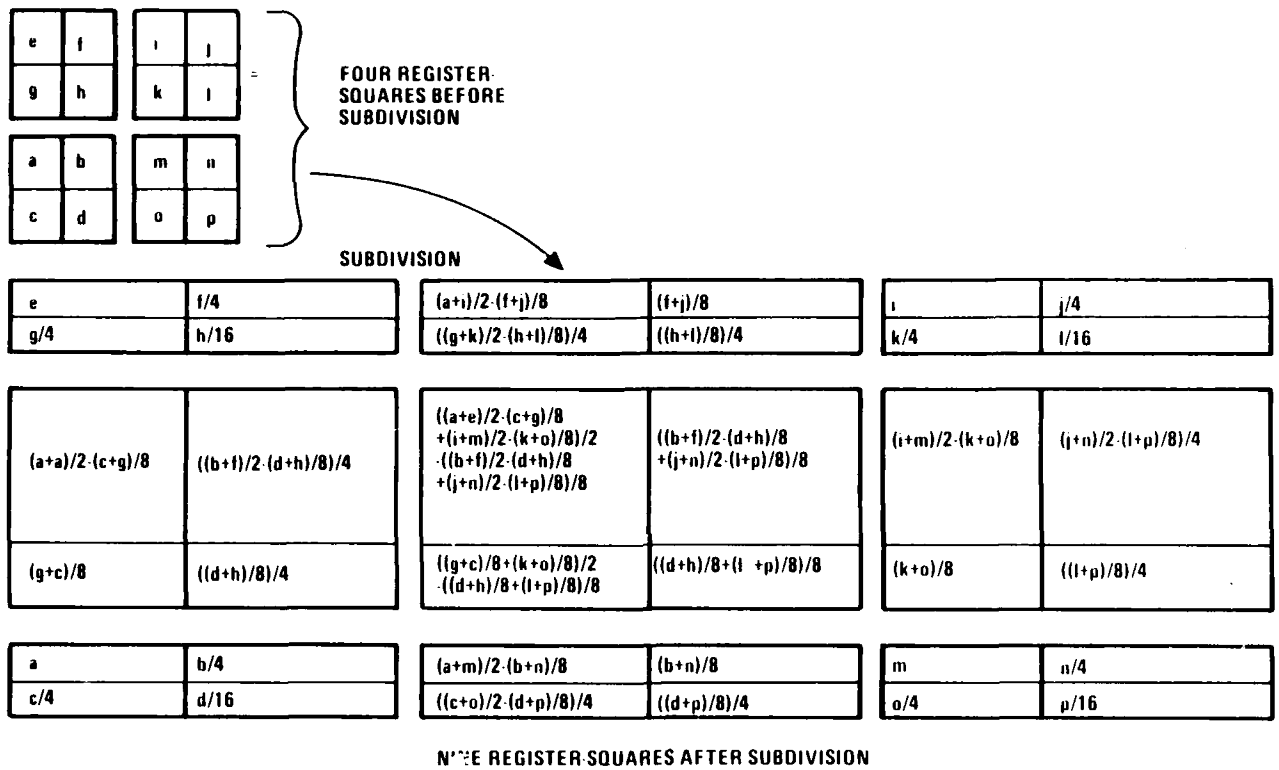


Figure 25. Subdividing register squares.

The result of any of these conditions is to terminate the scan at that node and discard all branches below the node at which termination occurred. We can immediately compute the proper intensity using the texture, shadow, and illumination models described elsewhere in this report.

Projected patches form polygons in screen space. These polygons will be enclosed by rectangles and the subpixels covered by each rectangle will be assigned the patch intensity. The approximation of coloring a screen by enclosing rectangles rather than by the polygons themselves has three justifications:

1. It allows the use of a compact, quick computation when determining exactly which subpixels are covered by a rectangle.

2. It avoids losing "bow tie" polygons; that is, if the surface normal calculated for a bow tie faces away from the observer, the polygon will not be displayed. Ordinarily, a hole will appear in the picture because of this error. By empolying enclosing rectangles, the overlap from rectangles adjacent to the bow tie will cover any potential holes (see Figures 26a and 26c).

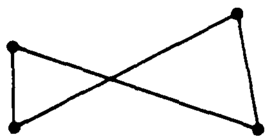
3. It avoids holes from the projection of adjacent patches which are subdivided to different levels (see Figure 26d).

Some time may be saved by comparing the relative lengths of polygon sides and subdividing only the longer dimension. For example, in Figure 26b if L1/L2 were greater than two, only the sides L1 and L3 would be subdivided. This process may also be desirable because by creating polygons of more or less equal dimensions on all sides, their enclosing rectangles will more accurately reflect the polygon dimensions; and less time will be spent replacing z-buffer memories because of reduced overlap among the rectangles. An investigation of this procedure in quantitative detail was not performed as part of this study, thus a recommendation of its use is left open.

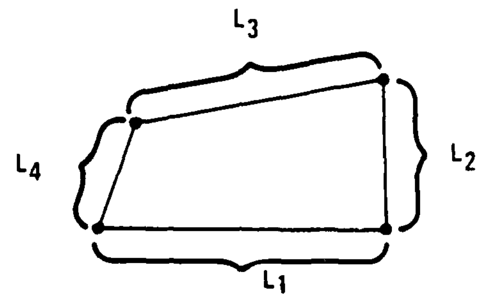
After a patch projection on the screen has been reduced to pixel size, it is tempting to try to save some time by determining those patches that face away and discarding them to avoid intensity calculation for those surfaces. This can be accomplished by taking the dot product of the "eye vector" with the surface normal vector of the patch. If the dot product is negative, the surface faces away from the observer.

Such a test was not implemented, for two reasons:

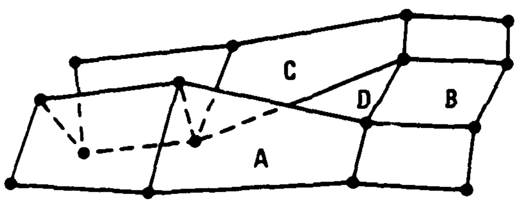
1. There are occasions when there are no surfaces facing away. In such cases the test is of no use and in fact becomes a liability. Thus the test is of no value in lessening the number of intensity calculations in the worst case: flat terrain filling the FOV out to the horizon.



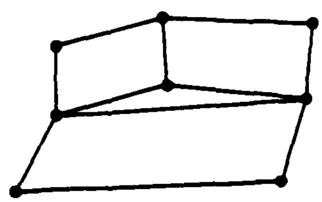
a. "Bow tie" Polygon



b. The patch corresponding to the above polygon will no longer be subdivided if $L_1 <$ pixel size, where only L_1 and L_2 are tested.



c. Rectangles enclosing A, B and C will cover the bowtie polygon D.



d. Rectangles enclosing these polygons will cover the triangular hole.

Figure 26. Patch anomalies.

2. Most terrain will be modeled with a diffuse reflectance term only, because it is perceptually adequate. Thus the specular term requiring the dot product of eye and surface normal vectors will be a rarity. It follows that the special computation of this dot product for surfaces facing away will be time-consuming, requiring a new normalized eye vector and dot product each frame time.

Figure 27 summarizes the subdivision and display algorithm.

5.2.3.4 Display Algorithm Benefits and Drawbacks--

Benefits--

1. Because bicubic B-spline surfaces represent the terrain surfaces, real-world contours are modeled more accurately than with polygons.
2. No sorting is required to determine patch maxima. This avoids numerical techniques which occasionally incur singularities.
3. Bicubic and fractal subdivisions are logical extensions of the mission quad tree subdivision. The transition from the stored quad tree to the computed quad tree can be transparent to the algorithm implementation. This provides a unified approach and a clear flow of logic from the root node to the smallest displayable subpatch.
4. Fractal subdivisions used for textures, texture borders, and shadow borders can be computed in parallel with bicubic subdivision, adding no time to the display algorithm.
5. The overall algorithmic approach is unified because subdivision is applied to obtain smooth curved surfaces, irregular textured surfaces, texture borders, shadow borders, and appropriate pixel size resolution for terrain and cultural objects.
6. Fractal texturing provides an arbitrary and automatic level of detail while maintaining the macroscopic identity of a texture.
7. A z-buffer is used to solve the hidden surface problem. Other techniques need to sort lists of entities to solve the hidden surface problem. The sorting can become a major factor in real-time considerations, limiting the number of polygons or patches that are sorted.
8. Each specialized technique can be flexibly implemented. Thus, shadows can be applied or withheld; cultural objects can be added or withheld; and textures may or may not be applied in any part or all of the data base.

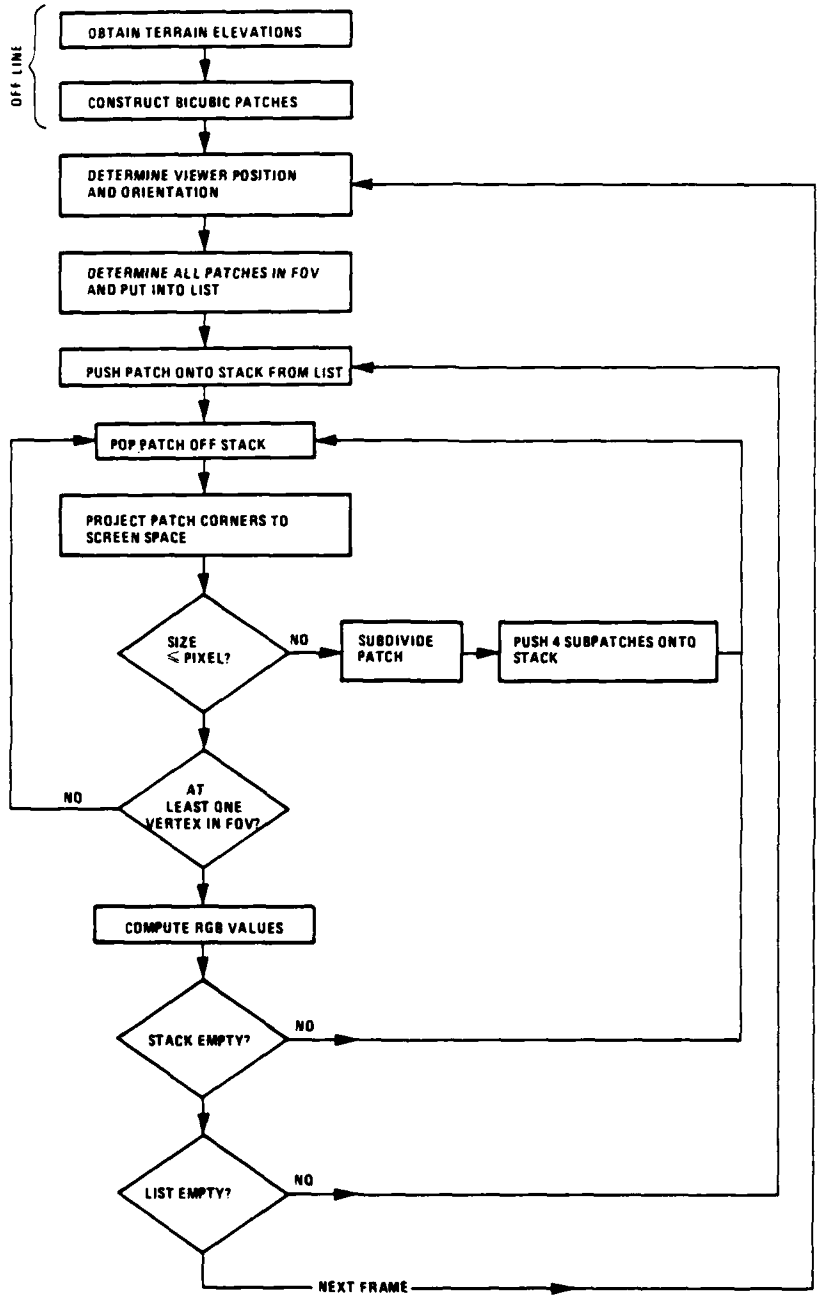


Figure 27. Subdivision and display algorithm.

9. Off-line preparation of the data base for register squares and texture is independent of the mission; that is, the preparation is independent of the sun position. Off-line preparation can be handled quickly because the affine transformation is already in part of the real-time perspective transformation hardware.

10. The nature of the subdivision process allows us to avoid the cross and dot products taken for the surface normal vector and diffuse intensity and employ instead a simpler add and shift process.

11. Bicubic and fractal subdivision processes are very quick, using only add, subtract, and shift operations.

Drawbacks--

1. Anti-aliasing is difficult because the surfaces which must be used in anti-aliasing become available at different times.

2. As a patch is subdivided, the space requirements grow rapidly for storing register squares. Adjacent patches with common corners must have separate register squares because the squares will contain values representing different levels of subdivision.

3. The cubic surface may not be appropriate for all surfaces. Flat surfaces may be rendered with slight undulations due to adjacent patches which have some curvature.

3.2.3.5 Hydrographic Surfaces--Bicubic patches form a continuous surface. When water surfaces meet land, however, there is generally a discontinuity. If this is determined to be perceptually important, an inexpensive method is available to model the discontinuity. For those patches containing water-land borders, two register squares can be stored, one for a flat water surface, the other for the terrain surface. In analogy with the texture border determination method, the elevation appropriate to the texture (water or land) can be assigned. Thus, in this special case, two bicubic surfaces are subdivided in parallel and the elevation chosen for display will be contingent upon the texture chosen for display.

3.2.3.6 Clipping--Clipping cultural objects or terrain surfaces not appearing in the FOV is accomplished in two separate stages during the subdivision process, and one just before subdivision. For each frame time, a circular area surrounding the viewer and contained in a high-speed memory is available. From this, a particular slice slightly greater than 60° is taken.

A typical slice might contain 50,000 patches; of these only 40,000 are subdivided. When a patch is projected to screen space for the first time, a rectangle surrounding the patch will be used for clipping. If any part of the rectangle intersects the screen, the patch is included in the rest of the subdivision and perspective transformation process. If no part intersects the screen, the patch is discarded. To accommodate highly-curved patches, the rectangle should be slightly larger than the borders defined by the patch vertexes.

When a patch is finally subdivided to pixel size, an intensity calculation is carried out only if the rectangle enclosing the patch can be assigned to a screen pixel; if not, it is excluded. Thus, the third stage of clipping is not a separate process but simply falls out of the process of assigning polygon intensities to screen pixels.

3.2.3.7 Terrain and Hydrographic Operations--There are two matrix multiplies which must occur in the off-line data preparation to generate each bicubic coefficient matrix. The operations for this are 128 multiplies and 96 adds. The bicubic coefficients must be used to generate register square values. This takes an additional nine multiplies and 40 adds. The total number of multiplies is 137 and the number of adds is 136. This number is linearly proportional to the size of the mission date base.

For a typical minicomputer a multiply time is 11.2 μsec and an add time is 0.38 μsec. Allowing one msec input/output time, the total time to generate one register square is 2.59 msec. With 5×10^6 register squares (the assumed mission area), 3.6 hours of computer time is required. In the on-line mode, the item of interest is the bicubic subdivision of each terrain patch. For this mission area, there will be 1.7×10^6 subdivisions per frame. This is accomplished using real-time pipelined hardware, as shown in Figure 5. See Appendix D for a discussion of other surface representation algorithms.

3.2.4 Object Representation

All the previous texture techniques treated texture as a pattern in which individual objects are not treated separately but as a collection. This allows the pattern to be represented in a more or less adequate and timely fashion when viewed from a distance. None of the previous techniques were appropriate for near-distance views. In this subsection, several methods are discussed for representing individual objects. The thesis is that a texture pattern is an aggregation of individual objects. As one comes nearer to a pattern, the pattern should break up into discernible objects. The original pattern may then actually disappear and be replaced by distinct objects. These have their own internal patterns that might be represented by the techniques previously described.

The techniques to be presented here are of two types. The first uses video object images. This is a version of mapping adapted to individual object representation. The second is object synthesis, in which well-known shapes like circles and rectangles are perturbed and textured to represent the internal patterns in the objects synthesized.

3.2.4.1 Images for Object Representation--Representation with images uses object images stored on video discs for superposition on surfaces. The idea merges the control and surface representation offered by traditional computer-generated imagery (CGI) techniques with the high fidelity offered by video technology. The speed with which a scene can be filled with high-fidelity objects is much faster than with traditional CGI object representation because the details of the objects do not have to be generated individually.

The control provided by surface specification and object placement in a uniform grid can be fruitfully combined with the high fidelity of object photographs and computer access to video discs. Objects can be positioned on the modeled terrain both manually and automatically. Photographs of objects can be stored and categorized according to type, aspect, size, and nearest point for display on video disc.

Once the computer has a list of object locations and sizes, the objects can be extracted automatically from the video disc. Note that the object to be extracted must be specified by size, distance from viewer, aspect, and type of object. Pictures can be stored of objects representing the closest distance from the viewer for which that picture is appropriate. Then, when a more long-range view is needed, the image can be digitally shrunk.

Object priority (hidden surface resolution) can be solved by using the z-buffer approach. This means that every pixel will have a distance and intensity associated with it. When an intensity is to be assigned to a pixel, the distance to the associated object is compared with the distance to the object already covering the pixel. If the new object is closer to the viewer, the intensity for that object replaces the old intensity at the pixel.

Illumination can be a bit difficult because surface normals cannot be determined at every pixel. In a forested area most pixels are contained in trees, bushes, grass, etc. Since these are being placed on the underlying surface from photographs, there is no control over the object surfaces. An option is to normalize the intensity distribution of each photograph as it is prepared for entry in the data base. The stored intensities then represent the ambient intensity, just as was done for mapping. Then, during scene construction, the intensity can be modified according to the scene content; for example, forest would be darker than fields. Also, in a forest bright blotches can be introduced to simulate the sun spots which sneak through small openings in natural forest canopies.

correct perspective can be achieved by using photographs from different aspects of the same object. During scene construction, the appropriate aspect to use can be determined by knowing the angle at which the line-of-sight intersects the object (depression angle) plus the distance from viewer to object. If a tree is assumed to be reasonably symmetrical, especially when viewed from a distance, there is no need to know the viewing angle around it.

To correctly represent an object as it is approached requires a sequence of photographs, each with greater levels of detail. To get continuity in the display image, a method used by cartoonists can be adopted. They produce key frames (every fourth frame is an original), then intervening frames are interpolated. The resampling method could be used to compute an image from the nearest available photograph. A resampled image from that farther photograph can also be produced. The displayed image would then be the average of the two. This should smooth the transition that would otherwise be obvious. Other variations with translucence and image interpolation are under current consideration.

Note that if the level-of-detail transition can be done with acceptable realism, the fidelity of the representation is as good as the film used to take the picture; and the object does not require a lot of computing time to construct it. However, the level of hardware integration is increased because video equipment must interface with digital equipment.

3.2.4.2 Object Synthesis--Models can be used to construct images of objects. The fidelity of the representation will, of course, depend on the quality of the model and the cost of using it. Some very good models of trees have been constructed by Marshall, Wilson, and Carlson (1980) but it takes several hours to produce one picture of one tree. Newell (1975) used polygons in procedure models to build objects and correctly portray them. Clark (1976) recommends object model generation by structuring to simplify the sorting problem.

Although some object models may need too much time to produce dynamically, models can be used to make pictures off-line. A data base can be built of video images which can be resampled for display in a scene.

An object generator may have access to other object generators in the fashion recommended by Clark (1976). A town generator may make requests from a road generator and a house generator and a tree generator. Any of these may use object images from image data base, or they may use any of the previously defined texturing methods for constructing objects. The following paragraphs describe how a generator for trees might be built.

A large proportion of natural terrain can be realistically modeled by two-dimensional textured surfaces during high-altitude flight. This is true for forests and for individual trees. The forests can be approximated by the "mottled" surface of their collective tree tops, and individual trees can be either ignored or treated as random spots on the earth's surface. However, during low-altitude flights, individual trees require specific models for realistic simulation.

Once one tree is constructed, this construction can be repeated to create a field of trees. By adding small variations at each repetition or by using a variety of trees (maple, oak, etc), realism can be enhanced. The major problem is how to create a single realistic tree at low cost (high speed). Since only the outside of a leafy tree is seen at a distance, the appearance of a given tree can be simulated by constructing a surface; there is no need to be concerned about the interior of the tree or its actual structure.

There are potentially several ways to model this surface, though it is not known which method is quickest or most realistic. First, the basic shape of a tree could be approximated with an ellipsoid of revolution, sphere, or cone and then the surface normals could be perturbed to create a textured appearance. Alternately, (with more computation), the surface itself could be perturbed. This would be more realistic because the tree silhouette would be irregular rather than smooth. In fact, this might be especially important for the recognition of evergreens in the winter since their dark green silhouette against white snow is more visually dominant than the contrasts among branches on the tree interior.

To perturb a surface, or its normal, LUTs might be used, or some sort of Markov process, or perhaps fractals. A choice could also be made between using real-world data or synthesizing our own. A determination of the best technique would require some experimentation.

Rather than perturb a surface or its normal, a textured surface could be mapped onto the ellipsoid, sphere, or whatever. This would be quick in comparison to perturbation but again silhouettes would be smooth. The tree's appearance would not change correctly as the observer moved with respect to the tree. (The surface itself would be perspectively correct as the viewer moved around it but the shading of the surface would be incorrect.) Since the tree surface would be smooth, spurious highlights would appear. It might therefore be necessary to simply assign a set of intensity values to the tree.

A particularly simple model would use three flat surfaces which mutually intersect, like two ellipsoids and one circle. This could be textured in a variety of ways but the important idea is to have the display algorithm ignore the internal edges and display only the silhouette. At certain viewing angles, parts of the planes might appear edge-on. If the display algorithm does not pick up this kind of high-frequency feature, it may not be a problem. Suppose that the planes were "noisy ellipses" instead of smooth ellipses. This would be easy to generate and much more realistic.

The easiest way to texture this model would be to assign a two-dimensional texture to the tree in image space. For example, the algorithm might be as follows:

1. Construct three-plane tree in object space.
2. Transform to image space, but draw silhouette only.
3. Texture inside of silhouette according to same statistical generator.

Note that this statistical pattern would remain the same regardless of viewing angle or distance. For trees at a good distance, this might be a good approach.

Other texturing procedures might include mapping a texture onto each plane or mapping a texture in image space which was dependent upon the tree orientation. In the former case realism would be sacrificed because, regardless of how irregular the texture was, it would look odd when a plane was viewed at a slight angle. The latter case is more interesting: Suppose that at step 3 it is known what direction the tree was facing with respect to the view (that is, a viewer vector and a tree vector are known), $\vec{A}$ and $\vec{B}$. Knowledge of the relative orientation of $\vec{A}$ and $\vec{B}$ (or perhaps just their dot product) would allow mapping of an orientation-dependent texture onto the tree interior. In this case, the tree's appearance would not change as Θ changed or as r changed but would as ϕ changed. The best texture parameter would alter the texture such that each incremental change in ϕ changed the tree texture realistically.

Benefits--Texturing techniques like mapping are useful when the scene is viewed from a distance. At short range the need for discernable individual objects increases. Mapping can be used but there are problems with perspective. Synthesized objects can be made to represent the amount of detail needed for effective training. An object generator can be constructed for every feature in the DMA feature list. Others can be added at will. The cost will of course depend on the amount of detail desired. For example, if trees can be adequately represented by triangles resting on rectangular trunks, the generator will be relatively cheap. If a lot of detail is needed, as in the method of Marshall, Wilson, and Carlson (1980), the cost will no doubt be high. The concept of synthesizing objects and representing textures as aggregates of objects is needed where high fidelity is required for effective training. Perhaps nap-of-the-earth flying or low-level bombing will have such requirements. The cost may then be justified.

Drawbacks--Object synthesis is in its infancy. No one knows what fidelity is needed for cost-effective training. As object synthesis matures, the methods should be simpler and experiments will be conducted to determine the contribution to training effect. At this time it is difficult to justify sophisticated object synthesis models. Simple models such as fractaled ellipses for trees, polygons for houses and bridges, and cylinders for oil tanks seem to be the proper level of use of this technique. All the DMA features can be represented by such models if object images prove inadequate but the cost in dollars and time may be excessive.

3.2.4.3 Curved Objects--In general, vector parametric bicubic equations are sufficient to generate a large class of objects with curved surfaces. The theory for these representations and examples of many objects generated from the representations are available in Blinn (1978); Catmull (1974); Newman and Sproull (1979); and Rogers and Adams (1976). A bicubic subdivision process is applied to these objects just as to terrain surface patches.

3.2.4.4 Operations--Under ordinary flight circumstances, the relatively small number of objects in an FOV will probably not add more than 10% to the number of surfaces requiring perspective transformations and subdivision.

3.2.5 Shadows

The realistic generation of shadows is an important factor in the simulation of terrain contours and cultural objects. Shadows are especially important as indicators of cultural object elevations besides providing clues as to their three-dimensional structure. The algorithm outline below describes the Honeywell approach to shadow generation, which has its origins in the algorithm of Williams (1978). However, the algorithm for generating terrain shadows is a Honeywell innovation.

I. There are several desirable or necessary features in a shadow algorithm:

   1. Use a Honeywell-Catmull type subdivision algorithm, rather than a scan-line algorithm, because of texturing and other advantages.
   2. Include shadows projected from objects lying outside the FOV.
   3. Small objects such as houses and trees should cast relatively sharp shadows.
   4. Far objects (clouds and distant mountains) should cast shadows which are fuzzy close up but sharp at a distance.
   5. When near surfaces shade distant surfaces, the distant shadows should be fairly sharp but should not be calculated using all of the foreground detail.

6. Off-line calculations should be incorporated to save time.

7. The shadow algorithm should be simple, quick, and should possess the potential for future improvements concerning unique shadow types.

II. Several factors should be avoided or minimized:

1. Long searches or sorts for sun-space hidden surfaces

2. Off-line calculations which determine shadows down to patch resolution, creating blocky shadows

3. Shadows cast by curved surfaces that are incorrectly calculated when rectangles are drawn about patch vertexes

4. Some surface patches which resemble bow ties and create ambiguities

5. Sharp shadow borders which create more aliasing problems

In this algorithm, five distinct types of shadows are considered; each type is treated differently.

- Type 1 shadows--shadows cast by stationary cultural objects (trees, houses, cars, etc). These shadows can be calculated off-line and stored as additional cultural features.
- Type 2 shadows--those occurring on stationary cultural surfaces facing away from the sun. Sun-space surface normal calculations (off-line) will determine these surfaces.
- Type 3 shadows--these are also called silhouette or terminator shadows. They occur only on patches containing sun space silhouettes; that is, only on patches in which one or more patch corner surface normals face the sun while the others face away, or a patch which is partially hidden from the sun by some other patch.
- Type 4 shadows--shadows that fall on those patches totally hidden from the sun by other patches. These shadows can also be tagged off-line.
- Type 5 shadows--those shadows cast by moving objects and computed during real time.

3.2.5.1 Algorithm Outline--Shadows may be generated for terrain, static, and moving cultural objects, either independently or in parallel. The shadows can be cast by the sun or moon from any sky position which is taken to be stationary throughout the mission simulation. The generation of terrain and static object shadows is accomplished partly off-line and partly on-line, whereas moving object shadows are generated entirely on-line.

Detailed Description--

1. Off-Line Preparation of Terrain Shadow Labels--The first step in simulating terrain shadows is to give each of the discrete DMA mission area elevations a shadow label, designating roughly whether each elevation terrain point is or is not in shadow. This is equivalent to determining which points are hidden from the sun/moon by some part of the terrain. This procedure is illustrated in Figure 28 and discussed below.

To avoid error, it is necessary to obtain the DMA elevations appropriate to the mission from one unified data base. This is because DMA files are arranged in $1^{\circ} \times 1^{\circ}$ square, and a mountain in one square might cast a shadow upon terrain in an adjacent square. Calculating shadows a square at a time (rather than for the whole mission data base) would leave unshaded regions along square borders, and this is unacceptable.

It is assumed that the sun/moon is located at infinity so that the shadows are cast by parallel rays. To determine whether an elevation point is hidden behind some terrain from the point of view of the sun, we employ a two-pass affine transformation (affine because the illumination rays are parallel).

In the first pass, four adjacent elevations representing the four corners of a patch are transformed at a time; this is done for all mission patches. Once these points arrive in "sun space," the sun space pixels covered by the patch are determined, and each covered pixel is assigned a depth (distance) equal to the average of the depths of the four patch corners. In addition, if a patch corner lies in one of the covered pixels, the pixel is re-assigned the depth value of the corner point.

The affine transformation is simply that given by the matrix transformation from world to eye coordinates, described in subsection 3.1.3 of this report. Pixel size in sun space would ideally be much smaller than the maximum projected dimensions of a typical patch, but storage constraints limit the pixels to those just small enough not to introduce distracting errors. Based on our experience in generating pictures with Catmull-like algorithms, it is assumed that four pixels per polygon (on the average) are required although future specific tests of the most appropriate sun-space pixel size are advised.

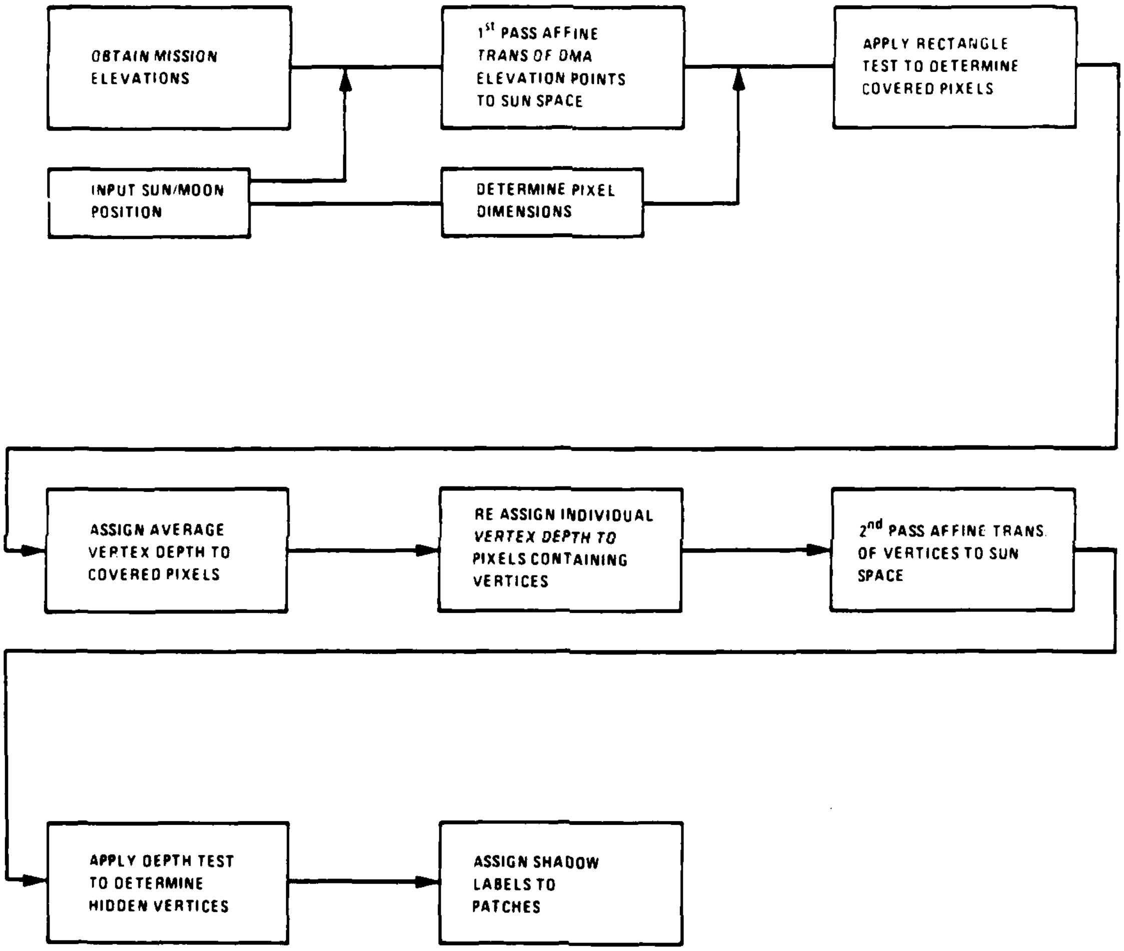


Figure 28. Off-line preparation of terrain shadow labels.

After an affine transformation, patch dimensions will be more or less equal over the entire sun-space "screen," allowing a uniform pixel size throughout. (Note: there is no physical sun-space screen--it is only a mathematical abstraction.) However, all vertical pixel dimensions will be scaled according to the altitude of the sun/moon, in proportion to the cosine of the altitude; that is, in proportion to the vertical dimensions of the patch projections on the sun-space screen. This ensures proper resolution by scaling pixels so that the ratio of their vertical to horizontal dimensions approximately equals the ratio of patch dimensions in sun space.

To determine which pixel centers are covered, either of two tests can be employed: the first approximates the patch projection with a rectangle enclosing the four vertexes. It is fast (22 adds per patch) but slightly inaccurate. The alternative approximates the patch projection with a polygon constructed from the patch vertexes. It is much slower (92 adds, 24 multiplies, 8 divides per patch) and more accurate. A singular defect of this alternative is that it occasionally misses so-called bow tie polygons (see Figure 26). Because of this potential error, and because the rectangle approximation is probably sufficient, the simpler and faster rectangle test is preferable. To accommodate those who might wish to test these alternatives in the future, a brief theory and computational estimate for both tests are provided in Figure 29.

After the sun-space pixels have been assigned depth values, a second-pass affine transformation of vertexes (elevation points) is made, exactly like the first. If a vertex falls within a pixel having a lesser depth value, the vertex is in shadow; otherwise, the vertex is *not in shadow*. It is convenient to transform four vertexes at a time so that the surface patch as a whole can be assigned a shadow label. If all four vertexes are in shadow, the patch is given a shadow label. If none of the four are in shadow, no label is assigned. If one, two, or three vertexes are in shadow, the patch is given a special "shadow border" designation and the specific shaded vertexes are also designated. Finally, each vertex that belongs to a shadow border patch is assigned a random integer "x," where $1 \leq x \leq 10$. It is essential that a vertex shared by adjacent patches have the same integer regardless of the patch that it belongs to. Figure 30 contains an estimate for the number of off-line operations performed in the preparation of shadow labels.

Rectangle Test Procedure

1. Given four vertexes in sun space, determine maximum and minimum x and y values (x, y are sun space screen coordinates).

2. Assign z-values to pixel centers inside the rectangle.

Total Operations: $\simeq$ 22

Total Time: 22 x 0.38 μsec x 5 x 10^6 patches = 42 sec

Polygon Test Procedure:

1. Given four vertexes, compute equation for each edge.

2. Using a clockwise (or counterclockwise) ordering rule, assign inequality rule denoting inside and outside to each edge equation.

3. Determine rectangular enclosure (as given above).

4. Select all pixel centers within the rectangle for testing by four inequalities and test.

5. Assign z-values to pixel centers within polygon.

Total Operations: 92 adds, 24 multiplies, 8 divides

Total Time: 92 x 0.38 μsec + 24 x 12.7 μsec + 8 x 14.6 μsec x 5 x 10^6
$\simeq$ 42 min

Figure 29. Rectangle and polygon tests.

There are several operations involved in obtaining shadow labels. By far the most time-consuming are the two affine transformations of the mission data base points. The remaining operations are insignificant in comparison, and can be ignored.

One affine transformation consists of a (4 x 4) matrix multiplied by a 4 vector. It requires at most 16 multiplies and 9 adds.

$$
\begin{array}{lcl}
16 * 11.2\,\mu s & = & 0.17 \text{ msec} \\
+\ 9 * 0.38\,\mu s & = & \text{negligible} \\
+\ \text{I/O avg.} & = & \underline{1.0 \text{ msec}} \\
 & & 1.19 \text{ msec per transformation}
\end{array}
$$

There are two passes of the transformations. Each point is redundantly transformed four times, and there are 5×10^6 data points.

$$\text{Total time} = 1.19 \times 10^{-3} \times 2 \times 4 \times 5 \times 10^6 = 13.2 \text{ hrs}$$

Note that special-purpose hardware used for the perspective transformation during realtime could be used for the affine shadow transformations off-line, thereby reducing the time considerably.

Figure 30. Off-line shadow label operations.

2. Real-Time Generation of Terrain Shadows--During the mission simulation, patches that have no shadow label will be left alone, and will be illuminated according to the method explained in subsection 3.2.1. Those patches with a shadow label are entirely shaded; that is, the intensity of the entire patch is made ambient appropriate to the constraints of weather and training effectiveness. In other words, the intensity is arbitrary and can be set at will.

The only patches requiring significant computation during real time are "shadow border" patches. These patches undergo fractal subdivision in the same way as is done for terrain texture. Under the best conditions (overcast sky or sun at 90° altitude) there will be no shadow border patches. However, when the sun (or moon) is low in the sky, the number of 100-meter square areas (patches) containing shadow borders may amount to a significant fraction of the total number of patches in the mission data base. For typical terrain, it is assumed that this fraction never exceeds 1/100, and is usually on the order of 1/500.

Shadow border patches are of three types:

1. One vertex in shadow
2. Two vertexes in shadow
3. Three vertexes in shadow

As with texture border patches, a "false" elevation of "1" is assigned to each vertex in the light and a "0" is assigned to each vertex in shadow off-line. Using the random numbers assigned off-line, and a single shadow fractal look-up table, a rough false surface is manufactured during the subdivision process. When the average false elevation of a subpatch is greater than one-half, that subpatch is assigned its ordinary reflectivity. When it is less than one-half, the subpatch reflectivity is decreased so as to shade it. Based on the operations estimate on texture, the operations necessary for terrain shadowing are:

Typical Case: 1/500 number of patches; fractal subdivision in parallel with all other subdivision processes (see Figure 19).

Worst Case: 1/100 number of patches; fractal subdivision in parallel with all other subdivision processes (see Figure 19).

It is important to understand that the off-line and real-time terrain shadow generation techniques are entirely independent of the other AVSS algorithms, and can be eliminated or included at will. Also, some future experimentation must be conducted to create the most suitable fractal LUT for shadow borders. It may even prove profitable to investigate the application of specific shadow LUTs for each terrain texture type, rather than just one.

3.2.5.2 Static Cultural Object Shadows--Cultural objects will be represented either as video images or as collections of polygonal and curved surfaces. Video image shadows can be obtained as additional video images. Thus a tree shadow would be stored as a video image along with its tree image. The shadow image distance would be assigned a priority bit so that, when placed at some surface location, the surface would not occlude the shadow.

Static cultural object shadows involving CIG surfaces are of two types: those occurring on cultural surfaces facing away from the sun and those occurring on surfaces cast by the cultural objects. For polygonal objects, an on-line dot product test of surface normals with the sun vector will determine whether a surface is or is not facing the sun. If not, such a surface may be assigned only an ambient intensity, and zero reflectivities, as are all surfaces in shadow.

Case shadows are assumed to fall on flat surfaces. This makes calculations much easier and is sufficiently accurate since locally the curved terrain surfaces are flat. Because the sun is in a constant position during the mission, all such cast shadows can be calculated off-line and stored. The easiest method to produce cast shadows is to project the object a polygon at a time onto the flat surface. Each projection of a polygon is then a shadow polygon itself. The collection of all shadow polygons from a single cultural object is the shadow of the cultural object.

During real-time image generation, these shadow polygons can be called up for display, and just as with ordinary polygons, they can be assigned positions in object space. Shadow polygons will also receive priority bits, so that when their Z_e and the surface Z_e are very close, the shadow Z_e will be assigned to the pixel.

Depending on the polygonal cultural object, some of its surfaces should not have projected shadows calculated. For instance, a flat-topped warehouse will produce shadows from only two of its side surfaces.

The display of polygons representing either cultural object or shadow surfaces is analogous to the display of terrain patch surfaces in that both are subdivided until they are pixel-size elements, at which point enclosing rectangles determines pixel coverage. However, cultural object polygons can be linearly subdivided in screen space very quickly; this is the method suggested for their display. Note that determining the pixel covered by cultural object polygons is not efficient if the enclosure test uses the polygon edges themselves (see Figure 29). This is especially true of the typically small polygons which compose a house or car, etc.

Curved objects will project curved shadows. These shadows are to be defined off-line for a given sun/moon position for each curved object. It is suggested that only simple curved forms be used (spheres, cylinders, cones) because their affine projections onto flat surfaces involve only straight line segments and ellipse segments. Thus, knowing the equation of the curved objects, the equation of their projections may be determined by projective geometry. From these equations, control points may be formed and thus cubic curves may be used to define the shadow boundary (see Rogers and Adams (1976); Chapters 4 and 5 provide details).

Moving objects should be modeled as collections of polygonal surfaces since curved surfaces make real-time shadow generation difficult. As with static cultural objects, moving-object shadows will be generated by an affine projection to a flat surface, one object polygon at a time but in real time. Assuming only one or two moving objects in an FOV, each composed of only a few dozen polygons, less than a hundred affine transformations would be required per frame time. The hardware for this transformation would be available because it is identical to part of the perspective transformation hardware (the matrix multiply). Consequently, cost and computation are minimal.

Advantages--It should be noted that terrain shadow generation is inexpensive because it employs the same bicubic methods off-line as are used for terrain patches. It also contributes to a unified scheme for the simulation of the real world and represents only a minor increase in complexity. Lastly, the vast majority of calculations are made off-line, reserving only the most necessary terminator calculations for real-time calculation.

Disadvantages--Because of the bicubic surface and the large changes in slope, the terminator may appear wavy regardless of how fuzzy it is. If this is the case, some special pre-processing may be necessary. For example, the shadow elevation(s) could be averaged with their neighbors.

3.2.6 Special Capabilities/Considerations

3.2.6.1 Lights--

Point Light Sources--Point light sources must appear as small as possible but also must not display aliasing artifacts. This object can be met by treating each point light source as an approximately pixel-size object with a Gaussian intensity distribution. Since the point source is defined to appear the same from all angles, the Gaussian distribution is independent of orientation and is a function of distance only. Figure 31 illustrates a typical point light source represented by three concentric boxes (pyramid) approximating a Gaussian distribution, superimposed on a grid of pixels. Note that the pyramid shifts translationally across the screen as the point source moves, but never rotates.

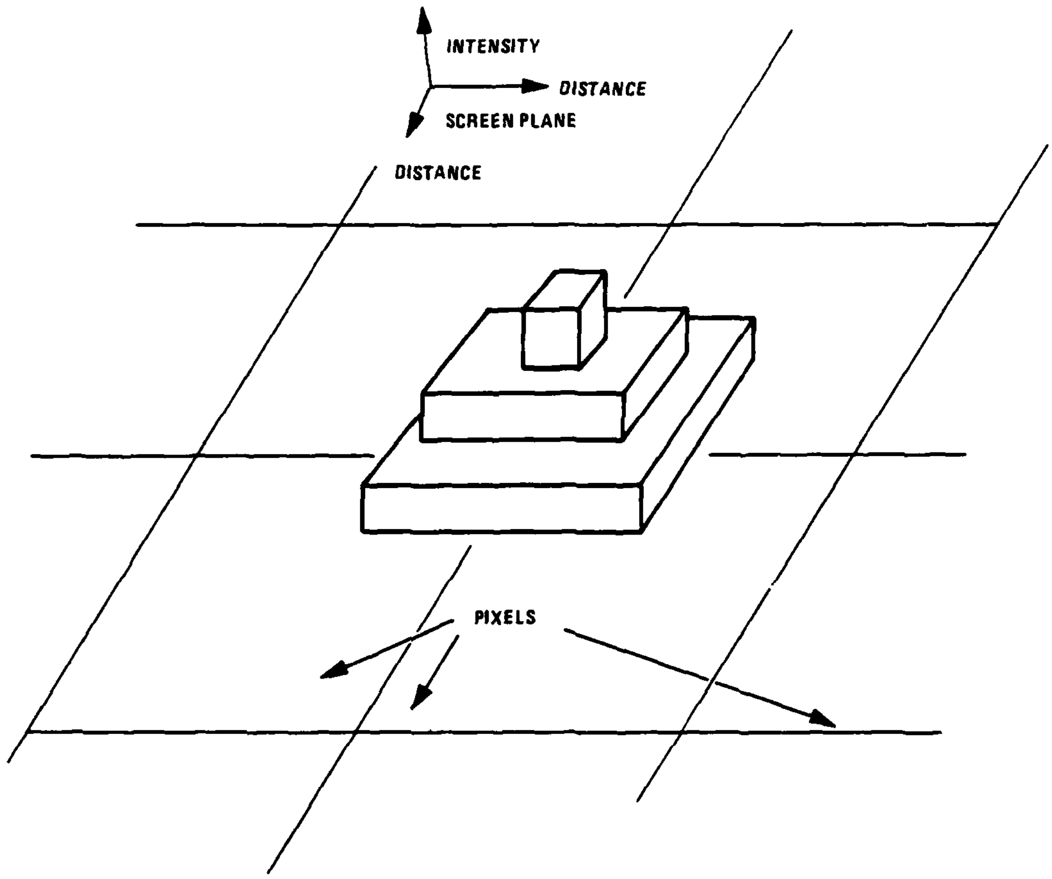


Figure 31. Point light source pyramid: approximation to gaussian distribution.

The exact pyramid dimensions must be specified after some perceptual tests, to minimize artifacts. The pyramid intensity decreases as one over the square of the distance ($1/d^2$). The intensity can be equated to the integral of the Gaussian. This integral is proportional to any of the box heights. For example, at one distance, the concentric box heights might be (5, 2, 1). At some closer distance they would be adjusted to, say, (10, 4, 2). Thus, the heights are all adjusted proportionally.

The distance used for the intensity computation is precisely

$$(X_e^2 + Y_e^2 + Z_e^2)^{1/2}$$

Again, however, perceptual tests may indicate that this factor can be approximated with a less costly function. Indeed, it is possible that the intensity function may be updated every few frame times without a noticeable flicker. The distance factor also appears in subsection 3.2.6.4. It is approximated there by Z_e, so the same convention is used here.

Occultation of point sources is best handled by treating just the occultation of the point itself, not the pixel-size object representing it. This reduces z-buffer compares and perspective transformations as well as more realistically providing an immediate onset or offset of the point source.

<u>Extended Light Sources</u>--Extended light sources are of two types: directional and *non-directional*. Non-directional (that is, isotropic) light sources are easily modeled by associating a constant intensity with the surface which serves as the light source structure. Directional light sources, like searchlights, can be modeled by a surface whose intensity depends on the dot product of its surface normal vector with the viewer vector in object space.

<u>Rotational and Flashing Capabilities</u>--Rotating a directional light source is simply a matter of making the surface normal of the light source rotate; that is, programming the vector normal position as a function of time. Similarly, the rotation of an object (such as a searchlight beam) can be pre-programmed. A flashing light will have its intensity programmed as a function of time.

3.2.6.2 <u>Translucence</u>--There are three categories of computer-generated translucence effects:

1. Atmospheric effects--includes rain, snow, haze, fog, etc.
2. Translucent objects--includes clouds, plumes from smokestacks, dust raised by vehicle movement or explosions, and smoke.

5. "Fade-in objects"--ordinary cultural objects that are designed to fade gradually into the scene by appearing initially as pixel size and highly translucent, with the translucence lessening as the objects move closer to the viewer. This special form of anti-aliasing helps avoid the distraction of objects which suddenly appear on the screen when they reach pixel dimensions. The alternative is to present cultural objects at a subpixel level, which is simpler but not as dependable because subpixel-size objects will not be accurately anti-aliased, perhaps leading to a "jumpy" appearance. Since an object is distracting only when it pops on the screen with a high contrast relative to the background, the special application of translucence should be of primary concern in the modeling of extended light sources at night.

In this case, the dark background makes the treatment especially simple. When greater than pixel size, an extended source can be assigned a constant intensity, and when less than pixel size, it can be treated as a point source, decreasing intensity as $1/r^2$, or more cheaply, $1/Z_e^2$.

The Basics--Given a scene which contains an object partially obscured by some medium, the following relation holds:

$$S(B_h) + \tau(B_o) = B$$

where

- B = brightness of the obscured object
- B_h = brightness of the obscuring medium
- B_o = brightness of the object without an obscurant
- τ = transmittance of light (how much light passes through the medium as a fraction of 1)
- S = saturation = $1 - \tau$, amount of absorbed or scattered light

This is called the translucence relation.

For example, given a bright object with $B_o = 150$ and a dark cloud ($B_h = 20$) that transmits 30 percent of the light incident upon it, the apparent brightness of the object will be:

$$(1 - 0.3)20 + (0.3)150 = 59$$

for an object at distance r, $\tau = e^{-\sigma r}$, where σ is called the "extinction coefficient" and represents the fraction of light lost when passing through some medium of thickness r.

The apparent brightness of an object viewed through the atmosphere or any other obscuring medium is (after substitution):

$$B = B_o e^{-\sigma r} + B_h(1 - e^{-\sigma r})$$

where

B = apparent brightness of the object viewed through a scattering medium

B_o = object brightness viewed at zero distance (or equivalently, with no scattering medium)

B_h = sky brightness

σ = fraction of incident light scattered per unit volume of medium

r = distance from viewer to object

This relation is the core of all translucence effects since it holds for all wavelengths and materials. In the end, therefore, only B_o, B_h, and τ, or equivalently, B_o, B_h, σ and r, are needed.

Derivation of translucence: When light is incident upon some small volume of a medium, part of the light is scattered equally in all directions, part passes through the medium.

The incident intensity is decreased in proportion to its magnitude; that is, a light twice as intense will loose twice as much to scattering:

$$\frac{dI}{dx} = -I\sigma$$

where dx is the thickness of the medium, σ is given above, and the minus sign is present because the intensity is decreasing.

The solution to this equation is

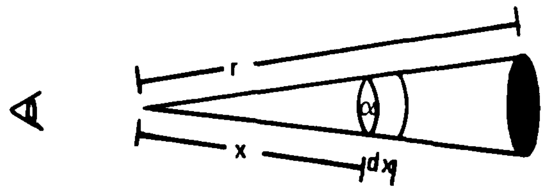
$$I = I_o e^{-\sigma x}$$

Now consider viewing a black object through the atmosphere (see Figure 32). The brightness of the black object can be found by first determining how much light is scattered from the cone volume into the eye.

A unit volume will scatter $I_o\sigma$; therefore, the light scattered in all directions by the volume element αdx will be $I_o\sigma\alpha dx$. This will be decreased by $e^{-\sigma x}$ as it goes through the medium. The solid angle of the viewing cone is α/x^2 (steradians).

Since there are 4π steradians for an entire sphere,

$$\frac{\alpha/x^2}{4\pi} = \frac{\text{light scattered into the eye}}{I_o \sigma\, \alpha dx e^{-\sigma x}}$$



αdx = VOLUME ELEMENT OF AIR (SCATTERING MEDIUM)
α = AREA OF VOLUME ELEMENT SEEN BY VIEWER

Figure 32. Viewing a black object.

Thus, the light scattered into the eye from the volume element

$$= \frac{I_o \sigma \alpha^2 dx e^{-\sigma x}}{4\pi x^2}$$

Since brightness is proportional to the light reaching the eye per surface area of the object, the brightness of the volume element will be proportional to:

$$\frac{1}{\alpha} \quad \frac{I_o \sigma \ \alpha^2 dx e^{-\sigma x}}{4\pi x^2} = \frac{I_o \ \sigma \alpha e^{-\sigma x} dx}{4\pi x^2} = \quad A\sigma e^{-\sigma x} dx$$

where A = constant, since α/x^2 = solid angle is a constant. Therefore, the total brightness of the air from the cone in front of the black object is:

$$\int_0^r A\sigma e^{-\sigma x} dx = A(1 - e^{-\sigma r})$$

The sky brightness, B_h, can be obtained by placing the black object at infinity:

$$B_h = \int_0^{\infty} A\sigma e^{-\sigma x} dx = A$$

With $B_h = A$, the brightness of the air in front of the black object is

$$B_h(1 - e^{-\sigma r})$$

For an arbitrary non-black object of brightness B_0 at zero distance, its brightness at distance r equals:

$$B = B_0 e^{-\sigma r} + B_h(1 - e^{-\sigma r})$$

Translucent Objects--The categories are:

- o Atmospheric clouds, all types that do not cover the entire sky and are structurally autonomous
- o Plumes of smoke or water vapor from factories or burning objects
- o Dust raised by moving vehicles and explosions
- o Exhaust from air vehicles
- o Smoke or spray from artillery or smoke pot generation. Exothermic, non-exothermic, instantaneous burn, finite burn, point source, linear array (of smoke pots or spray).

To model these objects usually requires a rough idea of the object dimensions as a function of time D(t), the brightness of the object B_h, (for example, white smoke or black smoke), and the transmittance τ, in some cases, as a function of time. In other words, it will be desirable to take into account the dependency of τ on the extinction coefficient σ and the optical depth of the object (r). However, some objects may be adequately treated by assuming a constant τ.

The algorithm for translucent object simulation assumes a base structure on which a fractaled surface is generated. The base structure can be altered in time to grow, elongate, or stretch in an arbitrary direction to simulate the expansion or diffusion of a cloud of dust or smoke.

The base structures can be stored or generated in a separate data base and each structure displayed independently. To simplify optical depth calculations for multiple surfaces, a cloud or puff of smoke can be modeled inward froman enclosing base structure (see Figure 33).

o Atmospheric Clouds--Modifications of fractal techniques should be powerful enough to model many cloud types. Beyond the simple base structure technique, some clouds can be modeled in the same way that Fournier modeled islands; intersecting a plane surface with a fractaled surface. By using non-Gaussian distributions, absolute values of distributions, and distortions in transformations, we will be able to surpass the confines of pure fractal simulations.

o Plumes of Smoke or Water Vapor--Plumes from factories vary enormously depending on literally dozens of factors. An accurate modeling is achievable with perhaps a month's effort. A crude modeling that can be done almost immediately simply employs a base structure resembling a curved truncated cone, which is fractaled to appear puffy.

  Translucence would be modeled as a function of the distance along the parametrically-defined plume. Thus, the cloud would disappear gradually.

  For IR views, a plume is often very bright at the smokestack, dimming at successively larger distances. For this effect, intensity can be changed as a function of distance.

o Dust, Exhaust, Smoke--Military and weapons effects are discussed in subsection 3.2.6.6; more details on smoke and water vapor translucence properties are contained in subsection 3.2.6.4.

LINE OF SIGHT THROUGH A CLOUD INTERSECTS SEVERAL SURFACES.

LINE OF SIGHT THROUGH AN ENCLOSING BASE STRUCTURE ALONE INTERSECTS ONLY TWO SURFACES. NOTE: THE BASE STRUCTURE IS NOT DISPLAYED; IT IS ONLY USED FOR AN OPTICAL DEPTH CALCULATION FOR VISIBLE PARTS OF THE CLOUD.

Figure 33. Modeling clouds for optical depth.

3.2.6.3 Moving Models--Moving objects are of two types: those with off-line, predetermined motion and those with free interactive motion.

Predetermined Aircraft Motion--A three-dimensional aircraft path can be described by a time-dependent position vector:

$$\vec{P}(t) = \vec{P}(x(t), y(t), z(t))$$

where x,y,z are the world coordinates and t = time = parameter, and by an aircraft orientation vector:

$$\vec{O}(x(t), y(t), z(t))$$

A precise knowledge of $\vec{O}$ depends on an explicit mathematical understanding of aircraft attitude for arbitrary paths, velocities, and accelerations, not to mention expected pilot maneuvers. Because of the complications inherent in specifying $\vec{O}$ and $\vec{P}$ for complex paths, predetermined aircraft motion should be applied only to the simplest of paths, such as circles and straight lines. In this way, $\vec{O}$ and $\vec{P}$ can be stored as simple vector parametric functions, rather than as a large collection of precomputed points, and the functions can be easily evaluated in real time.

In air-to-air combat scenarios, it is suggested that the enemy aircraft be represented symbolically. Because of the relative speeds involved, it is unnecessary to try to represent a small object with curved or flat surfaces. The orientation of flight surfaces can be represented by straight, narrow lines whose contrast with the background is a function of weather and perceptual conditions.

Predetermined Ground Vehicle Motion--Ground vehicle motion is quite different from aircraft motion because $\vec{P}$ and $\vec{O}$ must obey the constraints of the terrain contours. Rather than specifying x(t), y(t), z(t) for the path, it is easier to specify x(t) and y(t) and substitute the x and y values into the bicubic surface equation to determine z(t). Then (x, y, z) can be stored off-line for each frame position and called up during real time.

An arbitrary curved path over arbitrarily curved bicubic terrain patches can be constructed off-line as follows:

1. Draw (by hand) a rough curve over an (x,y) terrain map to indicate the vehicle path and starting and ending points; provide the vehicle speed.
2. Determine the intersection coordinates of the path curve with the (x,y) grid lines. These latitude-longitude grid lines correspond to the bicubic patch edges.
3. Substitute each coordinate pair into the corresponding bicubic surface equations to determine the intersection elevations (z).

4. Using the (x,y,z) coordinates as control points, construct a vector cubic spline so that each surface patch which contains part of the vehicle path has an associated vector cubic curve defined as

$$\vec{C}(p) = [x(p)\ y(p)\ z(p)]$$

where p = parameter, and x, y, z are all cubics in p.

The curve contained in each patch has endpoints at p = 0 and p = 1.

5. It is assumed that the vehicle will move at a constant velocity, that is, it will move equal arc-lengths each frame time. Off-line, the total arc-length of each patch curve will be determined by numerically integrating the following expression:

$$s = \text{arc length} = \int_0^1 +\sqrt{\left(\frac{dx}{dp}^2\right)+\left(\frac{dy}{dp}^2\right)+\left(\frac{dz}{dp}^2\right)}$$

6. With the velocity, determine the distance (Δs) moved in 1/30 second. Then determine $s/\Delta s$ = number of Δs intervals in s.

7. Eventually, (x, y, z) will be needed at arc-lengths of 0, Δs, $2\Delta s$, $3\Delta s$, $4\Delta s$...s, giving the vehicle position each frame time:

t = 0	1/30	2/30	3/30	
s = 0	Δs	$2\Delta s$	$3\Delta s$	etc

This gives x(p), y(p), z(p); but x(s), y(s), z(s) are needed.

However, the analytic relation between these expressions is difficult. Therefore, numeric methods will be employed.

8. Numeric method:

   a. Sample the parameter p at 0, 1/20, 2/20, 3/20,...19/20, 1. (The number 20 is arbitrary).

   b. Using the arc-length integral equation 20 times, find the arc length for each p-valued segment:

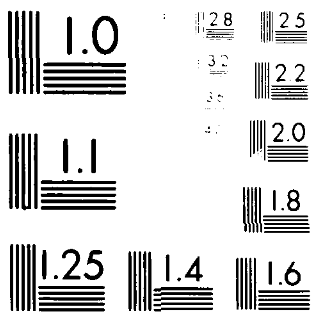
$$s(p = 0,\ p = 1/20,\ \text{etc.}) = \int_0^{1/20},\ \int_{1/20}^{2/20},\ \ldots\ \int_{19/20}^{1}$$

   c. Fit a curve to these points and then sample the curve at equal Δs intervals, where the number of samples per patch = $s/\Delta s$.

AD-A107 098 HONEYWELL SYSTEMS AND RESEARCH CENTER MINNEAPOLIS MN F/G 14/5
COMPUTER IMAGE GENERATION: ADVANCED VISUAL/SENSOR SIMULATION.(U)
OCT 81 D SERREYN, D DUNCAN F33615-80-C-0006
UNCLASSIFIED AFHRL-TP-81-23 NL

2 of 2
AD A

END
DATE
FILMED
12-81
DTIC



MICROCOPY RESOLUTION TEST CHART

d. Going backwards, each Δs will correspond to some p value, which in turn can be substituted into x(p), y(p). This gives the vehicle (x,y) position as a function of time (or equivalently, arc length). For each (x,y) pair, substitute into the bicubic patch equation to find (x,y,z). These values should be stored off-line as points.

Notes:

- o With a 60-Hz refresh rate, the same position is used for two consecutive frames. A one-minute simulation would require storing 60 x 30 = 1800 positions but a longer simulation is easily obtained by closing the predetermined vehicle path and looping back through the data.
- o To avoid having Δs intervals overlap patch borders, the nearest integer to $s/\Delta s$ can be used, giving only a slight velocity discontinuity at borders, for example, (40 mps $\rightarrow$ 40.08 mph).
- o The greatest amount of off-line expense is incurred by having to determine the path. Ostensibly, the computation of 20 integrals per patch seems time-consuming, but in step 5, the integral from zero to one can be composed of 20 rectangular sections, so that each of the integrals in 8b can be taken immediately as those rectangles; for example,

$$\int_0^{1/20} f(p)dp = f(1/20) \times 1/20$$

Vehicle Orientation--Vehicle orientation can be specified by two vectors: a surface normal vector $(\vec{N})$ and a vector tangent to the vehicle path $(\vec{V_t})$. Rather than take derivatives, these vectors can be approximated with no loss in realism.

$$\vec{V_t} = \frac{\vec{P}(x_{i+1}, y_{i+1}, z_{i+1}) - \vec{P}(x_i, y_i, z_i)}{\Delta s}$$

$$\vec{N} = \vec{V_t} \times \vec{W}$$

where $\vec{W}$ = vector tangent to the surface and more or less perpendicular to $\vec{V_t}$.

The base-point coordinates of $\vec{W}$ and $\vec{V_t}$ are equal. Without proof, the endpoint coordinates of $\vec{W}$ = (x,y,z), where specifically:

$$x = -\sqrt{25m^2/(m^2 + 1)} \qquad m = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

$$y = \sqrt{25/(m^2 + 1) + y_i}$$

The value of z can be found by substituting into the bicubic equation for the surface patch. (The factor of 25 is arbitrary and determines the length of $\vec{W}$, which should be short enough to be an accurate representation of a tangent rather than a secant but not so short that computer errors predominate.)

To summarize, for a vehicle moving over a given patch, position and orientation information will be calculated off-line and stored, for predetermined motion.

Interactive Vehicle Motion--Interactive aircraft motion consists of position and orientation being updated each frame time by an interactive control mechanism and operator. This information is fed to the screen space of the trainee and processed just as any other surface information: with intensity computed, z-buffer, and anti-aliasing applied.

In the real world, the orientation of another aircraft may be sighted at a great distance because the human eye is especially sensitive to linear features. In these cases, the simulation of such resolution is not obtainable by simply averaging the aircraft intensity over a pixel. There are two potential solutions.

The first, and simplest, is to model the distant aircraft with a several-pixel length "stick model" whose flight surfaces show an obvious orientation. For high-speed combat, this may well be adequate. We believe this model should be attempted first. The second solution involves a high-resolution projection system which would have to be specially designed and added to the flight simulator. This would incur considerably more expense and should be considered only as a last resort.

Interactive ground vehicle motion is constrained to the ground surface. The interactive control mechanism provides (x,y) position information directly each frame time or indirectly through velocity and direction information. Given the (x,y) position, the z value can be determined by substituting into the bicubic equation of the surface patch containing (x,y). To know which patch the vehicle is on, a search must be made through some predetermined list of patches, which is not too large. It follows that knowledge of the possible patches across which a vehicle will pass must be provided off-line before the mission is run, and that the bicubic equations must be known for each of these patches. The latter information can be obtained as soon as the bicubic coefficients for the patches are determined off-line (the bicubic equations are completely determined by the bicubic coefficients). It so happens that, with a bit of manipulation, the coefficients can be obtained from the register squares, so that a ground vehicle area may be determined after the mission data base is formed, this is just a less efficient procedure.

Vehicle orientation is adjusted each frame time by the method described in the previous subsection on predetermined ground vehicle orientation. This requires an additional substitution into the bicubic equation. Thus for each frame time, two substitutions must be made. These operations constitute the majority of computation for a vehicle traversing arbitrary terrain. The few dozen multiplies involved are trivial in comparison to the other system computation rates.

When fractal perturbations are added to terrain surfaces, they will be small enough to not differ from the bicubic elevations significantly where vehicle motion occurs. Thus, for example, a tank will not appear half-buried in the terrain, or floating above it. If some particular textural effect is a high priority and can be obtained only by using large fractal deviations, a special processor can be implemented to display intensities according to the perturbed surface normals while using unperturbed elevations for the surface display. This is analogous to Blinn's (1977) method. This alternative is available but is not solicited for implementation because of the additional cost and complexity.

3.2.6.4 Atmospheric Effects--Atmospheric visibility is modeled in accordance with the translucence relation:

$$B = B_o e^{-\sigma r} + B_h(1 - e^{-\sigma r})$$

where

B_h = sky brightness

B_o = object brightness with a clear sky

B = total brightness

r = distance between observer and object

σ = "extinction coefficient"; represents the fraction of light lost

The sky brightness, B, is calculated as follows. B_h and σ are determined off-line with σ depending on the weather. B_o and r are determined each frame time for each pixel corresponding to the terrain and cultural object data base. B is then found for each pixel each frame time. Some calculations may be avoided by employing an LUT containing $\tau = e^{-\sigma r}$ for various r and constant σ, which is constructed off-line and is weather-dependent. The table would contain 256 values of τ in equal increments, ensuring a continuous range of brightness.

Thus, the procedure is to obtain σ for a given mission and construct the LUT off-line. Then, during real time, consult the table for each polygon distance (r), find τ, and plug into the translucence equation to find the final object brightness.

σ, or equivalently τ for various r, is available (Duncan, 1980) for conditions of haze under all humidities, sea and land fogs, light, medium, and heavy rain, and snow for the following bandwidths:

Visual = 0.4 to 0.7 µm

LLLTV = 0.8 to 1.1 µm

FLIR = 8 to 14 µm; (3 to 5 µm sometimes)

For additional realism, the sky itself can be modeled as hazy along the horizon with video techniques. For example, a simple sky photograph would be used as a background for all computer-generated imagery. It would only have to be rotated and translated in real time to accommodate all perspective scenarios. Note that this technique is especially realistic and can even handle night skies with the stars appearing in their proper locations.

Altitude-dependent visibility effects can be modeled to any of four levels of detail. The first and crudest simply ignores such effects and assumes a homogeneous isotropic atmosphere. The fourth level incorporates both isotropy and homogeneity effects, and the second and third levels incorporate one of the two effects.

Consider, for example, that mountain peaks generally appear clearer than mountain bases. This can be simulated by substituting a fictional observer-object distance (r_f) into the translucence equation, where

$r_f = r - g(Z_w)$

$g(Z_w)$ = some function of the object elevation; depends on sensor and weather

The above is a particular isotropy model.

Now, consider that an observer on the ground would use a different r_f than an observer at some arbitrary altitude. Thus, a more realistic (and expensive) view would result in

$$r_f = r - g(z_{wo}, z_{wp})$$

where

Z_{wo} = object elevation

Z_{wp} = pilot elevation

Clouds--The merits of two types of cloud models will be discussed. However, no one method is indicated as best because future tests are required to accurately determine their technical feasibility, cost, and perceptual qualities.

1. As an extension of terrain texturing, fractals suggest themselves for clouds. For example, a puffy appearance may be generated by using absolute values in a fractal LUT. This can be superimposed as a perturbation on ellipsoids to generate cumulus clouds, or upon a flat surface to generate an entire bank of clouds.

   Alternately, imagine creating a fractal surface on a flat surface of large extent. If the perturbations are arranged correctly, some will be above the flat base surface, some below. By cutting out those below and assigning a uniform white to those areas above, a partly-cloudy sky surface can be manufactured. This is potentially realistic and it would employ the same hardware used for terrain fractals. Unfortunately, it would be time-consuming because of the mass of additional perspective transformations incurred. This could be reduced by taking advantage of the coherence of the flat, cloudy sky surface and subdividing in image space; however, special image space hardware would then be required.

2. Video methods show some promise in making both individual clouds and cloudy sky surfaces. The former can be treated by taking static photos of the clouds required for any weather scenario and placing them in the mission data base. The perceptual approximation of keeping the same cloud face toward a pilot flying by may be sufficiently real to employ. (A video tape of trees passing by with the same face always to the viewer has been examined and appears to be acceptable). With this approximation, only screen translations, rotations, and magnifications would be needed.

   With a cloudy sky surface, an additional operation of "perspective magnification" would be needed. This is where closer parts are magnified more than distant parts; it is equivalent to a perspective transformation. An algorithm for this process is discussed in Catmull and Smith (1980). This algorithm would probably require special real-time hardware. The great advantage of video clouds is their realism, their applicability to all cloud types, and the ease with which they may be made translucent, since they are simply a video overlay of terrain or sky imagery.

3.2.6.5 Sun/Moon--Modeling the sun itself is accomplished by creating a uniformly bright circle and placing it at a constant orientation with respect to the world. Similarly, the moon and various phases of it can be created and appropriately placed.

During the days of low irradiance (overcast skies), the illumination is partly isotropic and partly direct (Stenger, Dungan, and Reynolds, 1979; Steven and Unsworth, 1979) and can be approximately modeled by adjusting the ambient intensity. The moon's illumination of the cloudless earth at night is insignificant unless the moon is gibbous or larger. In such cases, terrain surfaces can be modeled dimly in a manner analogous to the sun illumination scheme. The only difference is that all surfaces will be made colorless since low light level human sight is insensitive to color. Color effects on terrain at sunset are relatively easy to add, simply shift the relative amounts of hue toward the red.

3.2.6.6 Weapon Effects--Explosive effects, flashes of colored light, etc, can be modeled as a succession of objects with each object lasting one frame time and the entire explosion lasting no more than 1/10 second. Each surface of the object would be assigned a specific intensity and would be independent of the usual object-intensity calculation. Therefore, each explosion would be perspectively correct and connected to a specific point on a surface element. It would be visible only if no terrain or cultural surfaces occluded it.

Smoke and dust clouds, whether generated by smoke pots, grenades, artillery bombardment, or fire, can be simulated by translucent, fractaled objects which grow or elongate according to wind conditions. Making the puffy smoke appearance change in time can be accomplished by using a time-dependent fractal LUT.

Object change can perhaps be modeled by video techniques, as can certain objects, but computer-generated damaged cultural objects can also be manufactured. Simple crater or bomb blast effects would be among the list of objects representing weapons damage where, depending on the object or terrain struck by a projectile, a given damaged object would be placed in the display. Again, the state of the art in video imagery does not allow us to predict the feasibility or expense of its implementation.

The great difficulty in simulating weapon effects is not that they are difficult to simulate, but rather that an overabundance of reference material is available on particular effects and the time spent in researching several dozen special effects for a detailed simulation is rather costly. See Zirkind (1978) and Ebersole and Vaglio-Lauvin (1980).

3.3 IMAGE QUALITY

To this point operational requirements and imagery characteristics have been covered. The third part of this section deals with the image quality or the elimination of known artifacts which occur in present CIG edge-based systems, to be considered are anti-aliasing and image complexity. In anti-aliasing, the concern is with quantization and interlace effects.

5.5.1 Anti-Aliasing

An ordinary raster image representation suffers from aliasing defects because it displays high-frequency features via a lower-frequency point sampling technique. When objects move across an image, further defects are introduced, such as the "jaggies" and the flickering of small objects which are sometimes lost by the sampling process. A rather elaborate analysis of aliasing effects may be found in Crow (1977).

Among the variety of anti-aliasing techniques which may be applied are blurring the image, sampling and displaying at a higher resolution, (convolving the image with a sine(x) function (or approximation), dithering, and area averaging. Blurring the image is cheap because one can simply defocus the display device, but it is perceptually inadmissible. Employing a higher-resolution sampling frequency and display is very expensive, does not cure all static aliasing artifacts, and is ineffective in treating temporal flicker and scintillation effects (Leler, 1980; Crow, 1977). Convolving the image is perceptually adequate but expensive. Dithering techniques rely on the temporal averaging properties of the human visual system by applying two simultaneous operations:

1. The projected surface elements on the viewer's screen are sampled at slightly different screen (x,y) coordinates each frame time. These differences never depart from one set screen position more than a pixel width.
2. The display hardware shifts each pixel on the screen to the sampling position used in the previous operation (1, above) each frame time.

If this procedure is carried out quickly enough, and in an appropriate manner over a pixel-size area, an anti-aliased image will be built up because the viewer tends to average quick, successive imagery.

Current display hardware allows a maximum sampling frequency of 60 Hz. Suppose object intensities were sampled at only two different screen locations and dithered corresponding to the opposite corners of a stationary pixel. The intensity at the first corner would be determined at $t = 0$, the intensity at the second corner at $t = 1/60$, back to the first corner at $t = 1/30$, etc. No flicker would be seen because the images fuse when presented at greater than about 25 Hz. However, the sampling would be inadequate because only two samples are taken. Aliasing would be reduced but not eliminated.

Sampling at four screen locations would reduce aliasing even more, but then a given sample point would be refreshed every 1/15 sec. In cases where a high-contrast object covered only one sample point, a distinctly perceivable flicker would be seen. Therefore, regardless of the number or distribution of sample points, dithering is not a feasible option.

The anti-aliasing technique chosen here is called area averaging. In its most rigorous form, it is equivalent to the convolution of an image intensity with some window (rectangular or Barlett triangular). However, it can be approximated by averaging discrete sample intensities over a pixel-size area.

An additional perceptual benefit of this technique is that apparent resolution will be increased. This is because below a certain angular size, object size and brightness are interchangeable; that is, they are perceptually equivalent. Leler (1980) discusses this in the summary of his paper.

3.3.1.1 Anti-Aliasing: Local or Global?--One of the most intensively studied issues in this study has been whether to apply anti-aliasing locally; that is, to certain small areas on the screen, or globally; to the entire screen. To apply local anti-aliasing one must know which areas require it. In general, these areas contain high-contrast features such as silhouettes, texture borders, shadow borders, cultural objects, and point sources.

Cultural objects, texture, and shadow borders can be localized and treated according to Duncan (1980). However, terrain silhouettes are very difficult to treat, especially when fractal perturbations are added.

Now suppose those pixels requiring anti-aliasing were easy to locate. If so, a dynamic memory allocation would be required. For scenes containing many high-contrast features (for example, low-level flight over bumpy terrain or any flight over shaded terrain when the sun is at a low angle, or high-altitude flight over diverse agricultural areas) as much as 10% to 20% of the screen would have to be anti-aliased. Clearly, the cost of such dynamic allocation would exceed simpler static global anti-aliasing requirements.

Therefore, because of the worst-case requirements, because there is no quick, straightforward localization technique for silhouettes, and because each high-contrast feature would require its own specialized localizing technique, global anti-aliasing is preferred. Neither should the potential importance of global anti-aliasing be underestimated for producing a uniform and homogeneous image of perceptually high resolution, as proposed by Leler (1980).

Crow (1981) investigated many different samplings for anti-aliasing. A 5 x 5 Bartlett window allowed no aliasing artifacts and a 3 x 3 Bartlett window showed slight artifacts with long, narrow, high-contrast specular features. Tests by others at Honeywell reveal that aliasing predominates at the along linear features but is not especially notable along irregular borders or in highly-textured areas. For natural terrain and ordinary cultural objects (houses, roads, etc), a 3 x 3 Bartlett window will suffice. This is because the predominant long, narrow, linear features are road surfaces, railroad tracks, and agricultural boundaries, none of which is of especially high contrast.

The window given by Crow is:

1	2	1
2	4	2
1	2	1

These are the weightings assigned to each sample point intensity. The weightings are very helpful in reducing aliasing. Typically, a non-weighted rectangular window is not as effective in aliasing reduction (Crow, 1981, Feibush, Levoy, and Cook, 1980).

Recall that in the display algorithm, pixel-size polygonal surface elements are approximated by rectangles. In general, these rectangles will have a maximum dimension equal to a pixel and a minimum dimension somewhat less. The enclosing rectangles are always oriented with sides parallel to the screen; that is, they are never rotated. Thus in the case where a polygon edge coveres one of the "2" weights in Crow's window, the polygon will be approximated by a rectangle which covers at least one of the "1" weights. This redundancy can be avoided by using a window like:

2		2
	4	
2		2

which has no side weights.

Since a 60-Hz frame rate is assumed, and since 30 Hz is greater than the fusion frequency for temporally flickering lights, the above can be applied by sampling three samples each frame time. At t = 0, the upper left, middle, and lower right samples would be taken and averaged. At t = 1/60, the upper right, middle, and lower left samples would be taken and averaged, and so forth, switching diagonals each frame time. This takes advantage of the eye's temporal averaging and consequently reduces storage and computation.

Small polygons can sometimes be missed by the sampling process and so effect a scintillation of the polygon as it moves. This effect is not present in terrain surfaces represented by many small polygons, but is in special cases like cultural objects that are of fractional pixel dimensions and of high contrast. The problem can be solved by setting horizon limitations on such objects. For example, houses would not be introduced until pixel size, and even then they would be blended into the image by applying a Z_e-dependent translucence.

Interlace is not discussed in detail since a non-interlaced raster scan has been assumed according to the requirements of the contract study objectives:

Summary Procedure:	5 point average/pixel
	3 points/frame
Operations:	$(1024)^2 \times 3$/frame time

3.3.2 Image Complexity

The Honeywell AVSS simulator is designed to maintain a fast memory FOV within a radius of 24 km, and to access disc memory to update the fast memory each frame time. Thus, the only information available to the simulator at a given instant is that information needed for the current FOV and the next few frames of potential FOVs. Because of the nature of the subdivision algorithms employed, a minimum of information is stored in the fast memory at a 100-meter resolution, from which an image containing arbitrary detail may be calculated each frame time.

The subdivision and testing methodology inherently maintains the correct amount of scene resolution under arbitrary circumstances. Transitions from one level of detail to another will not be perceptible because a maximal pixel dimension is maintained for all projected terrain and cultural surface elements (patches or polygons) and these elements are anti-aliased.

3.4 DATA BASE GENERATION

The use of two data bases is proposed. The first is the full detail of the DMA (1977) source data, including texture data, terrain and hydrographic data, and man-made features. This data base needs to be enhanced and reorganized for real-time scene generation. The data base produced by reorganizing and enhancing is the second data base, called the mission data base. The following paragraphs describe the procedure for enhancing and reorganizing the DMA data base to form the mission data base.

The mission data base is constructed from the world data base, as shown in Figure 34. Only the terrain likely to be viewed during the training mission is copied to the mission data base. If the training scenario starts with the student at a particular altitude and distance, the flight path can be defined as being anywhere within a corridor. The mission data base will then contain the terrain and cultural data needed to display scenes anywhere within the corridor.

This data base is automatically formatted in a curved-earth coordinate system because the world data are stored in the World Geodetic System. The mission data base allows for modification of individual features through the procedures for texture synthesis and for individual object representation during scene construction.

Part of the procedure in constructing the mission data base is the production of regional data blocks. A regional data block is formed from 16 known elevations and contains all cultural features within a 100-meter square. The four elevations and the surrounding 12 elevations are used to compute register squares. The data block contains pointers to the lists of objects, linear features, and texture areas representing cultural data in the block (shown in Figure 35).

3.4.1 Texture Region Representation

The digital land mass system (DLMS) data base has coordinates for points on the border of regions of homogeneous texture; that is, regions with one SMC code. The coordinates are in latitude and longitude offsets from a manuscript reference point. Texture regions can be constructed from the boundary coordinates by first converting the boundary representation to chain code, then forming and organizing blocks in a quad tree.

The DMA states that the border definition is not guaranteed to close if the region border extends beyond the manuscript edge. In this case closure will have to be forced by following the manuscript edge and creating the missing coordinates.

The DLMS cultural feature files contain codes defining the type and distribution of objects within a region. Distribution for a region can be used to place objects in the region. To simplify the process, a minimal surrounding rectangle can be defined around the region. This is a rectangle which touches the extreme ends of the region. Object allocation within the rectangle puts an upper limit on the number of objects while simplifying the border definition. As each object is assigned to a latitude and longitude position, it can be passed through the quad tree to see if it belongs to a data block. If the quad tree scan fails, the object is in the rectangle but outside

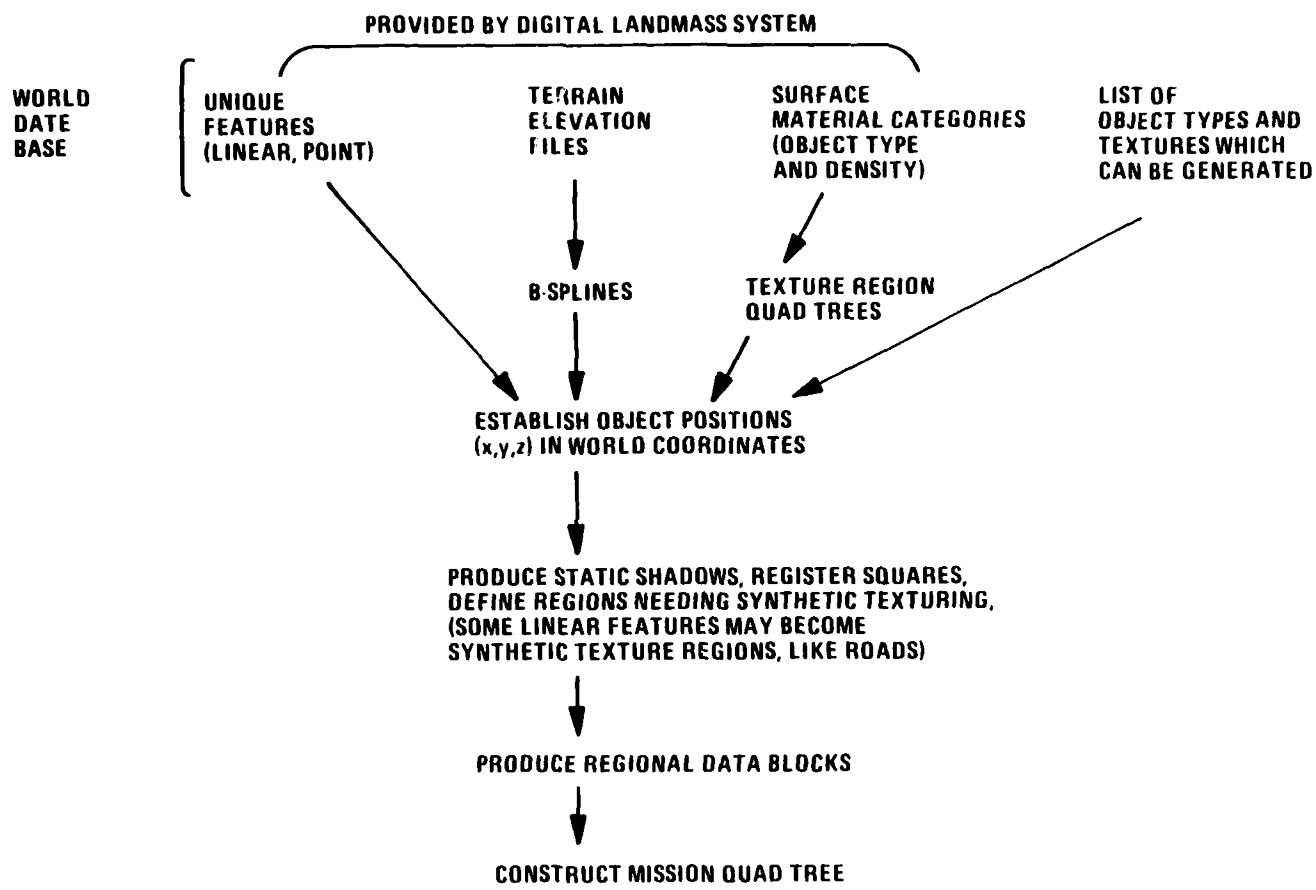


Figure 34. Mission data base construction-procedure.

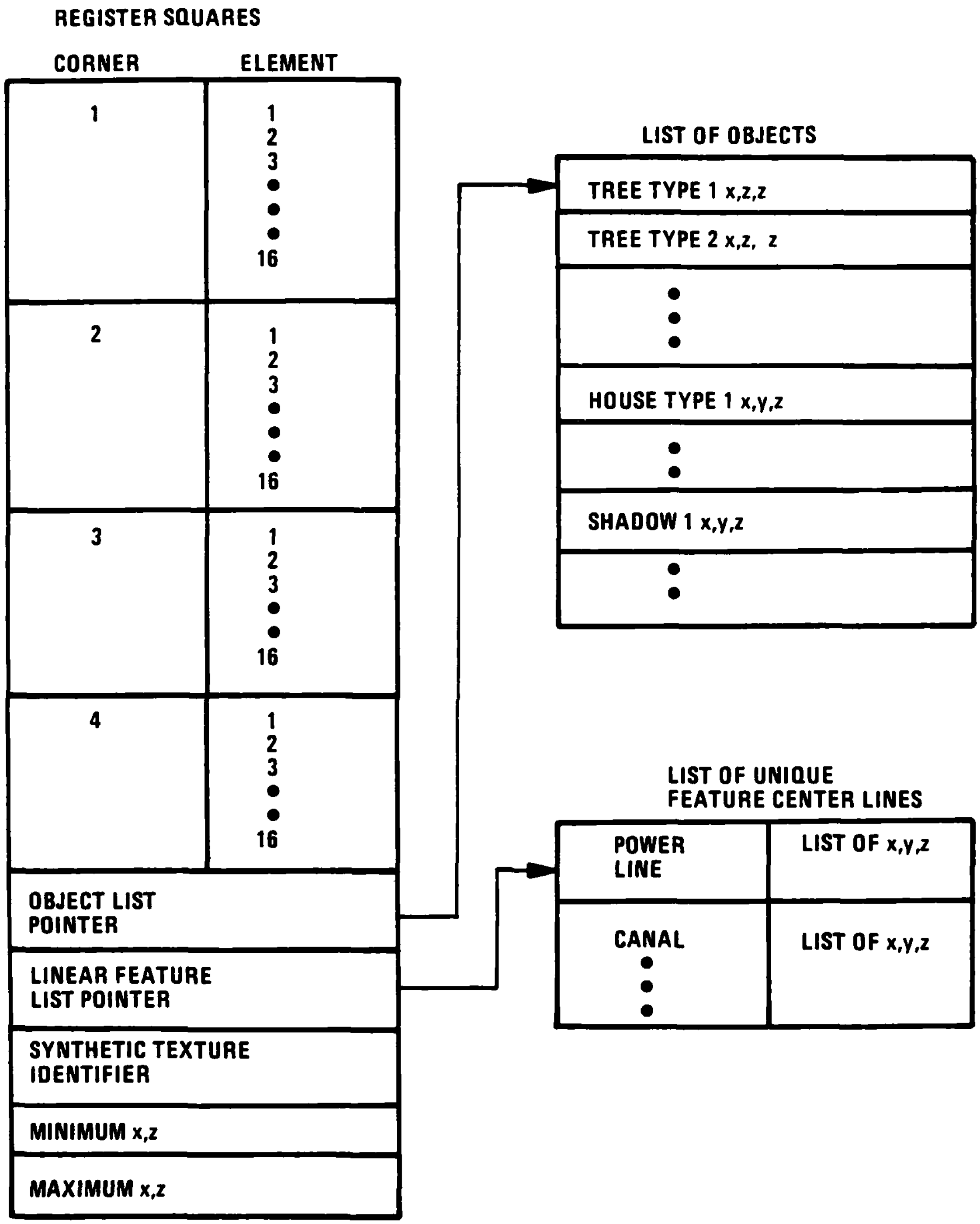


Figure 35. Regional data block.

the region boundary. When the object does fall inside a data block, the object's elevation is computed from the elevation coefficients. The object is then added to the block's object list.

Object placement is accomplished with a model for each SMC code. At most 13 models are needed, one for each SMC code. Each model would have access to feature models corresponding to the feature identification codes. The percentage of tree and roof coverage plus feature-specific information can be used to distribute objects of the appropriate type over the area. The list of objects produced by the model can be sorted in latitude and longitude to speed up the allocation to regional data blocks. Notice the objects are not being generated at this time. Only the object type and position are produced.

As a summary, the procedure for mission data base construction is:

- o Step 1--Define the mission gaming area.
- o Step 2--Identify the terrain and cultural data files included in the gaming area.
- o Step 3--Provide a list of the object types and textures which can be generated.
- o Step 4--Produce texture region quad trees.
- o Step 5--Compute bicubic elevation surface coefficients at one-second intervals.
- o Step 6--Merge results of steps 4 and 5 to make regional data blocks.
- o Step 7--Establish object positions.
- o Step 8--Construct object lists for data blocks.
- o Step 9--Set pointers to unique features and synthetic textures for each data block. Select texturing method based on SMC:
  --Mapping
  --Fractals
  --Aggregation of individual objects
- o Step 10--Define illumination conditions (sun or moon position, visible or IR, weather). Add IR characteristics to object descriptors.
- o Step 11--Produce static shadows.
- o Step 12--Construct register squares.
- o Step 13--Construct mission quad tree from all regional data blocks.

The output of this procedure is shown in Figure 36. The mission data base is the one used to produce scenes in real time. Conceptually the data are organized as they appear in Figure 37.

3.4.2 Mission Data Base Enhancement for Illumination

The mission data base at this stage is adequate for producing black and white images for the visible spectrum. To produce color images or infrared images, we need to add information to the data base. The following paragraphs will describe the procedures for making color, LLLTV, and IR images. The data base enhancements required are listed in Table 1.

3.4.2.1 Color--Use of the HSV color space is proposed. This color space is represented as a cylinder. The axis up the center of the cylinder corresponds to intensity in black and white. Black is at the bottom of the cylinder, white is at the top. Hue is what the layman calls color, such as red or green. Hue is determined by the angular position around the central axis of the cylinder. Saturation is the distance from the central axis toward the outer surface. This is illustrated in Figure 38.

Mission Data Base:

- **Is the collection of all regional data blocks (now called patches)**
- **Contains in each rdb**
  1. **Register squares defining bicubic elevation surface**
  2. **A pointer to list of unique features which pass through the patch**
  3. **A pointer to: A synthetic texture definition and a list of objects**
  4. **An intensity for display when the patch projects to one pixel**
- **Has patches representing variable-size areas**
- **Is organized as a quad tree to facilitate finding the patches for a given frame and controlling the level of detail processed**

Figure 36. Mission data base contents.

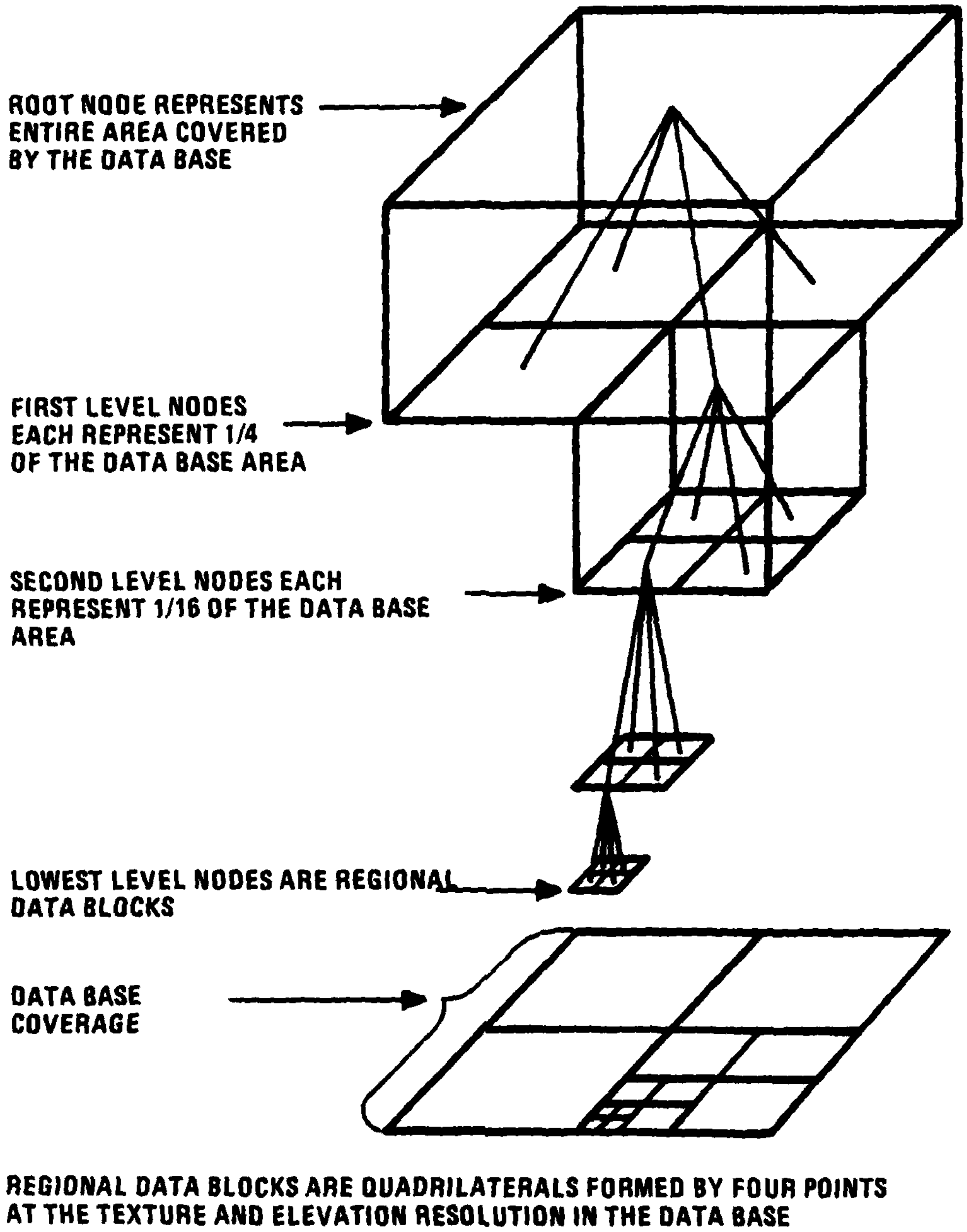


Figure 37. Quad tree data base organization.

TABLE 1. DATA BASE ENHANCEMENTS FOR ILLUMINATION

FLIR	LLLTV
Surface material --Solar absorptivity --Ground/sky absorptivity --Sensor band absorptivity --Surface cross-section	Sensor band reflectivity Surface condition Internal lighting conditions for a feature Point spread function
Internal thermal environment for a feature	Response curve
Sky thermal radiance	<u>Color for Visual</u>
Ground thermal radiance	Hue
Air temperature	Saturation
Wind vector	Shininess coefficient for specular component
Attenuation rate	
Precipitation	<u>All the Above</u>
Point spread function	Sensor band
Response curve	Sun position
	Sun direct radiance
	Sun diffuse radiance

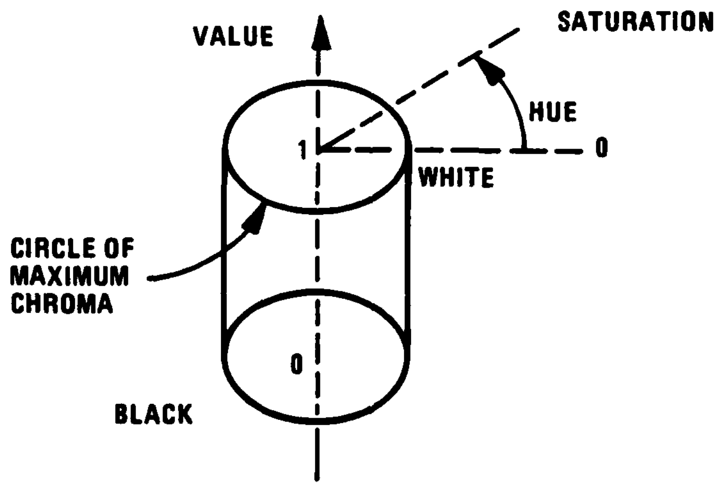


Figure 38. HSV color space.

One advantage of the HSV space is that the effect of atmospheric scattering can be simulated by interpolating between the object's hue and a nonsaturated blue. This produces the blue overtone seen on objects viewed at a distance. Another advantage is that interpolation can easily preserve saturation between hues, while this is more difficult in other color spaces.

A three-dimensional table is needed to use HSV. The indexes to the table are H, S, and V. The values in the table are the colors to be produced by the color guns in the display. Value (in HSV) is determined by the illumination model. Value is the intensity computed by the illumination model. Hue and saturation are object-dependent, so they must be specified in the data base or during object generation. This means each texturing method and object generators must have access to and include hue and saturation parameters.

3.4.2.2 Infrared--The illumination model for the FLIR sensor band (8 to 14 μm) is substantially different from the visible light. To compute the intensity on a surface, whether it is natural terrain or the surface of a cultural object, we need to know the pattern that corresponds to the temperature and emissivity of the surface. Even if the emissivity of the surface can be assumed to be static, the temperature is not. It depends on the time of the day, the direct radiance from the sun, the diffuse radiance from the sun, and other factors.

Thermal modeling of the surfaces of cultural objects, specifically buildings, has been done with the view of computing the temperature profiles of buildings. The profiles include the reflection of the sun, air, windspeed, and direction as well as precipitation (Zimmerlin, Suthy, and Stenger, 1979). This model attempts to predict the temperature of the surface based on the characteristics of the surface materials, which include the solar absorptivity, ground/sky absorptivity, sensor band absorptivity, surface cross-section, and internal thermal environment. These characteristics plus emissivity of the surface must be added to the mission data base.

3.4.2.3 LLLTV--The low light level systems in use today are the silicon vidicon and its derivations and fiber-optically coupled single-diode, double-diode, or micro-channel image intensifier systems. The simplest device is the silicon vidicon, which has a response extending into the near-infrared, especially when used with tungsten lighting. An object is seen against its background because there is a reflectivity difference causing contrast. The wavelength dependence of the contrast is determined by the materials involved. However, no one material can be characterized by a single reflectivity curve, so materials are usually characterized by their average reflectivity.

In summary, Table 1 lists the enhancements needed for generation of LLLTV, FLIR, and color images. In addition to Zimmerlin, Suthy, and Stenger (1974), Shanu (1975) and Thomson (1865) also provide useful information.

5.4.3 Data Base Size/Content

The mission data base is the collection of all regional data blocks (patches). Each regional data block contains:

- o Register squares defining bicubic elevation surface
- o Pointers to lists of cultural features
- o Pointers to fractal look-up tables (shadow borders, texture borders, textures)
- o A surface normal vector
- o Pointer to textural shadow information

These data are organized as a quad tree to facilitate finding the patches for a given frame.

The amount of data present is based on a gaming area of 800 km x 48 km. As an estimate there will be 5×10^6 data elevation points to be considered. Each elevation data point has the following bits associated with it:

o	Vector normal	48 bits
o	X, Y location	32
o	Size	16
o	Texture	8
o	Shadow	8
o	R^2 Data	256 (16 x 16)
		368 bits

For all patches, the data storage is:

$$\frac{5 \times 10^6 \cdot 366}{8} = 230 \text{ Mbytes}$$

For a safety factor, another mass storage unit can be added, to give a total capacity of about 512 Mbytes.

SECTION 4

IMPLEMENTATION CONSIDERATIONS

Implementation requirements for the on-line real-time processing are discussed in this section, including the fast FOV memory, perspective transform, testing, subdivision, intensity calculation and occultation, anti-aliasing, and the frame buffer. Each of these areas requires a special implementation based on the amount of data to be processed, how fast the hardware can be expected to operate, and the specific algorithms to be implemented.

4.1 HARDWARE IMPLEMENTATION

A previous discussion considered the ramifications of having a narrow FOV versus an entire FOV. The conclusion was that for best results the entire 360^o about a particular position should be stored so that transport delay does not cause a problem. For a 60^o FOV about 2 Mbytes of static RAM are required, for the full 360^o 12 Mbytes are needed. This memory is organized as 240,000 by 368 bits, with only 40,000 being accessed at any one frame time.

The access time for these memories is about 400 nsecs, which is well within the dynamic RAM capability of today's technology. The data in these RAMs must be organized such that simulator position input is easily converted to the RAM address.

In Table 2 the estimated number of 64K x 1 dynamic RAMs is 1500. The function of Table 1 will be to estimate the complexity of the hardware to be implemented. The next function to be evaluated is the perspective transformation.

The perspective transform was shown previously in Figure 2 during the discussion of transport delay. In this function, the data in world space must be projected onto the screen. This involves a subtraction, matrix multiply, addition, and then division for the X_e, Y_e values. In Table 1 the number of parts for this function are again estimated. The parts are commercially-available transistor-transistor logic, large-scale integrated (TTL LSI) chips, except for possibly the divider. The divider will probably be an LUT which gives the reciprocal (1/x), followed by a multiplier.

The delay for TTL parts worst-case for one block is assumed to be 200 nsec. In the case of the perspective transfer, this would apply to the multiplier. One could also assume that some of these items would be implemented using emitter-coupled logic (ECL) technology. For the ECL approach, a 25-nsec delay might be assumed.

TABLE 2. PARTS COUNT FOR PIPELINED APPROACH

Function	16-Bit Adders	16-Bit Subtractors	16 x 16 Bit Multipliers*	16 x 16 Bit Dividers	16-Bit Latches	16-Bit Multi-stage Shift Registers	Memory (RAM)*	Total Units
FOV Memory (12 Mbyte)							1500 (64K x 1)	1500
Perspective Projection (80 channels)	6 per channel	3 per channel	9 per channel	2 per channel	4 per channel			1920
Subdivision Stage							4000 (64K x 1)	4000
Patch Subdivide (20 channels)	21 per channel	10 per channel			9 per channel	32 per channel		1440
Grid Test (80 channels)	34 per channel							2720
Compute Intensity (60 channels)	7 per channel	5 per channel	18 per channel		10 per channel	47 per channel	3 per channel	5400
Z-Buffer Depth (10 Mbyte)							1250 (64K x 1)	1250
Intensity (20 Mbyte)							2500 (64K x 1)	2500
							Total	20,730

*Available today.

The effects on computational performance can be examined by looking at the throughput rate for this block. Using a 200-nsec stage delay, the number of operations that can be performed is five million. The number of perspective transforms is estimated to be 6.8×10^6 per frame or 408×10^6 per second. If the throughput is 5 MHz for the bipolar case then slightly over 80 parallel channels of this hardware will be required. On the other hand, if the throughput is 40 MHz for the ECL approach then only 10 parallel channels are required.

The test function shown in Figures 3 and 4 requires the same 6.8×10^6 tests, so again the number of channels is 10 for the ECL approach and 80 for the bipolar approach.

For the subdivision (Figures 5 through 8), only one-fourth the number of subdivisions is required, or 1.7×10^6. In this case the number of channels would be three for the ECL case and 20 for the bipolar case. Note that the perspective transform and tester should logically be four times as many as the subdivision for maximum throughput since each subdivision produces four new values at the end of the subdivision process.

For the intensity calculation and the occultation (shown in Figure 9), both of which occur in parallel, the number of points requiring an intensity calculation would be about 5×10^6 values. This allows for some of the 6.8×10^6 values to be discarded before the intensity calculation. This portion would then require about 60 channels of the bipolar technology approach, or about eight channels of an ECL variety.

Two channels of intensity memory are required so that while one frame is being calculated, the other is being displayed.

The overall computational efficiency cannot really be measured or compared with present edge-based systems. It is possible with some new techniques for edge-based systems to go between 50,000 and 400,000 edges. Just how good this is perceptually and how it relates to the system approach is unknown. The edge approach is estimated to take about 27,000 integrated circuits (ICs), whereas the estimated number of ICs is about 41,000. The justification for this count is based on the report by Soland, Voth, and Narendra (1979). In the report, approximately 8000 ICs were estimated. Applying the factor for the number of channels previously estimated to the parts estimated gives a total count of 20,730 as shown in Table 2. A safety factor of two for control circuitry is applied to get the 41,000.

The parts described in Table 2 are either available or can be available in hybrid packages. For instance, today there are no 16-bit adders or subtractors commercially available. However, the five chips required for the adder/subtractor can be combined into a single hybrid package if the project would warrant. The 16-bit multiplier is available today, as are the 64K x 1 memory chips. The 16-bit latches can be made from two 8-bit latches. Also, the shift registers can be made with hybrid techniques. The only parts not readily available are the 16-bit dividers. The units described in Table 2 assume that the hybridization has been completed for the aforementioned functions.

These ICs would take up approximately two racks, each 19 inches wide by 7 feet high, excluding the computer system in our system concept. This amount of equipment is significantly less than present-day systems.

In terms of impact of additional windows, the primary changes take place after the FOV memory. Hence the hardware that must be duplicated would be all the transformation, test subdivision hardware. A similar argument exists for two different eyepoints. However, if the difference between eyepoints is not great, it might be possible to determine an error amount and simply adjust the data that appears at the first eyepoint. This could be investigated further.

In the edge-based system, the sort must be on the polygon structure. As this structure gets larger, the amount of time required for sorting does not increase linearly with the number of polygons. Hence a significant amount of time would be taken up simply in sorting the data.

4.2 FUTURE TRENDS

This section has concentrated thus far on using the parts that are commercially available in the bipolar TTL LSI and ECL LSI chips, or hybrids from those chips. Another approach is to use MOS techniques and go to very large integrated circuits (VLIC). Clark (1980) is already proposing a chip he describes as the Geometry Engine. This chip does three functions which are common to computer graphics: transformation, clipping, and scaling.

The system concept proposed by Honeywell is computationally intensive in the transformation. Such technology could significantly reduce the amount of hardware that would have to be developed. Honeywell is pursuing the development of chips aimed at providing the computational efficiency required for AVSS as well as other optical systems. Briefly, the chip will do parallel arithmetic operations on a matrix of data. This is the parallel pipeline architecture required in AVSS to provide the realism lacking in today's training simulators.

SECTION 5

CONCLUSIONS

This report has provided the results of an investigation into candidate techniques for an AVSS. The system concept as a result of this investigation is based on the Air Force's needs in simulation.

Current techniques display polygonal approximations to terrain surfaces. Such displays contain little or no texture, no shadows, and few special effects. The low fidelity of polygonal displays is effective in transferring certain skills, like air-to-ground weapons delivery, but is ineffective for low-level or nap-of-the-earth flight training (Needham, Edwards, and Prather; 1980).

The AVSS system will be able to address most, if not all, of the important training aspects required of the next generation of flight simulators. Specifically, the system algorithm is based on high-fidelity terrain, texture, and shadow representations; and includes the capability for various special effects.

Terrain is represented as a smooth, continuous surface. Both its shape and illumination will more accurately represent true terrain than polygons. Terrain texture will be simulated by fractals, which are especially suitable for training in low-level flight, because the texture smoothly changes in level of detail for arbitrary distances while maintaining the macroscopic identity of the texture. Furthermore, the texture is three-dimensional; so the texture appearance is illumination-dependent, and it can be applied to model forest roofs and other irregular terrain features during low-level flight.

There is some perceptual evidence that the divergence in the optical flow field of terrain texture is an important distance cue (Koenderink and van Doorn, 1976). There is also evidence that local information (for example, changes in textural details) is processed by the human visual system for purposes of determining direction and velocity of motion (Regan and Beverly; 1978, 1979). Thus, detailed texture may be central to the successful simulation of the real world and to training tasks where accurate distance and direction cues are needed.

The AVSS system provides for the generation of terrain and object shadows, both important for terrain avoidance and object identification. Interactive and preprogrammed moving vehicles can also have shadows. Since the hardware for shadow generation is identical to part of the perspective transformation hardware, little extra cost is incurred when many moving vehicle shadows are required.

Translucence effects may be modeled in the system by applying different weights to the intensities of the occluding and hidden surfaces at a given pixel, and so introduces little extra cost.

To reduce image complexity to a minimum, an artificial horizon is used in the system. Only those surface elements within the horizon are displayed. The system concept is described quantitatively with an assumed 24-km maximum radius horizon. For special cases of flat terrain and low-level flight, or low visibility, the horizon may be arbitrarily reduced. Small cultural objects are displayed only when sufficiently large (pixel size).

Full-screen anti-aliasing is included, to smooth the imagery and eliminate "jaggies," flickering, and other disturbing artifacts. It should also be noted that this system presents a minimum of straight edges, which generate the most severe aliasing artifacts. In other words, current systems maximize their aliasing problems because they display straight edges of polygons. The AVSS system represents shadow terminators, texture borders, and terrain by smooth or irregular edges, and so inherently minimizes aliasing. The cost of full-screen anti-aliasing is fairly high but no adequate alternative exists if fractal textures are used. For reasons stated above, the potential training advantages of using fractals were felt to outweigh all other concerns.

The advantages and disadvantages of the display algorithm are summarized as follows.

5.1 ADVANTAGES

1. Because bicubic B-spline surfaces represent the terrain surfaces, real-world contours are modeled more accurately than with polygons.
2. No sorting is required to determine patch maxima. This avoids numerical techniques which occasionally incur singularities.
3. Bicubic and fractal subdivisions are logical extensions of the mission quad tree subdivision. The transition from the stored quad tree to the computed quad tree can be transparent to the algorithm implementation. This provides a unified approach and a clear flow of logic from the root node to the smallest displayable subpatch.
4. Fractal subdivisions used for textures, texture borders, and shadow borders can be computed in parallel with bicubic subdivision, adding no time to the display algorithm.

5. The overall algorithmic approach is unified because subdivision is applied to obtain smooth curved surfaces, irregular-textured surfaces, texture borders, shadow borders, and appropriate pixel size resolution for terrain and cultural objects.

6. Fractal texturing provides an arbitrary and automatic level of detail while maintaining the macroscopic identity of a texture. It is the best available simulation technique for low-level flight scenarios.

7. A z-buffer is used to solve the hidden surface problem. Other techniques need to sort lists of entities to solve the hidden surface problem. The sorting can become a major factor in real-time considerations, limiting the number of polygons or patches sorted.

8. Each specialized technique can be flexibly implemented. Thus, shadows can be applied or withheld; cultural objects can be added or withheld; and textures may or may not be applied in any part or all of the data base.

9. Off-line preparation of the data base for register squares and texture is independent of the mission; that is, the preparation is independent of the sun position. Off-line preparation can be handled quickly because the affine transformation is already in part of the real-time perspective transformation hardware.

10. The nature of the subdivision process allows avoidance of the cross and dot products taken for the surface normal vector and diffuse intensity; instead, a simpler add and shift process is used.

11. Bicubic and fractal subdivision processes are very quick, using only add, subtract, and shift operations.

5.2 DISADVANTAGES

1. Anti-aliasing is difficult because the surfaces which must be used in anti-aliasing become available at different times.

2. As a patch is subdivided, the space requirements grow rapidly for storing register squares. Adjacent patches with common corners must have separate register squares because the squares will contain values representing different levels of subdivision.

3. The cubic surface may not be appropriate for all surfaces. Flat surfaces may be rendered with slight undulations due to adjacent patches which have some curvature.

In order to address these drawbacks, the following are recommended: The texture approach chosen does provide the necessary realism. Textures should be generated and evaluated over a predetermined flight path with several different textures. Then this sequence should be shown to pilots to verify that perceptual cues are indeed obtained from these texture surfaces. Anti-aliasing will be applied over the image. During the simulation, an estimate can be made of how long it takes the various surfaces to be available.

Second, the simulation will help to determine how much intermediate storage is required during the subdivision. A significant number of memory chips has already been allocated for the function; it is desirable to reduce this component count.

Third, the induced curvature to real, smooth surfaces caused by adjacent real patches which do have slight curvature needs to be evaluated.

In general, before committing to a full-scale hardware development, certain areas should be simulated and evaluated. A preliminary design might be a part of the simulation. The results will be positive for the texture approach chosen.

REFERENCES

Bajcsy, R., Friedman, N. and Sloan, K.R., "Aerial Perspective--Monocular Depth Cue for Landscape Scenes," IEEE Joint Workshop on Pattern Recognition and Artificial Intelligence, June 1976.

Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," Computer Graphics, Vol. 11, No. 2, 1977.

Blinn, J.F., "Computer Display of Curved Surfaces," Doctoral dissertation, University of Utah, December 1978.

Carpenter, L.G., "Computer Rendering of Fractal Curves and Surfaces," Siggraph '80, Special Issue, July 1980.

Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces," Doctoral dissertation, University of Utah, 1974.

Catmull, E. and Smith, A.R., "3-D Transformations of Images in Scan Line Order," ACM Siggraph '80 Conference Proceedings, July 1980, pp. 279-285.

Clark, J., "Hierarchical Geometric Models for Visible Surface Algorithms," CACM, Vol. 9, No. 10, October 1976.

Clark, J., "A VLSI Geometry Processor for Graphics," Computer, July 1980, pp. 59-69.

Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," CACM, Vol. 2, No. 11, November 1977.

Crow, F.C., "A Comparison of Anti-Aliasing Techniques," IEEE Computer Graphics and Applications, Vol. 1, No. 1, January 1981, pp. 40-49.

Defense Mapping Agency Product Specifications for Digital Landmass System (DLMS) Data Base (1st Edition). Defense Mapping Agency Aerospace Center, St. Louis AFS, MO 63118 (PS/ICD/100; PS/ICE/100; PS/ICF/100; PS/ICG/100), July 1977.

Duncan, D., Honeywell Interoffice Correspondence, Honeywell Systems and Research Center, 2600 Ridgway Parkway, Minneapolis, MN, Mail Station MN17-2306, 19 August 1980.

Duncan, D., Honeywell Interoffice Correspondence, Honeywell Systems and Research Center, Honeywell Inc. 2600 Ridgway Parkway, PO Box 312, Mail Station MN17-2306, Minneapolis, MN 55440, September 30, 1980.

Ebersole, J.F. and Vaglio-Lauvin, R., "A Comparative Assessment of Battlefield-Induced Contaminant Degradation of Atmospheric Transmission at IR and MM Wavelengths," Aerodyne Research, Inc., Center for Electro-Optical Signature Studies, Crosby Drive, Bedford, MA, May 1980.

Feibush, E.A., Levoy, M. and Cook, R.L., "Syntactic Texturing Using Digital Filters," Siggraph '80, Vol. 14, No. 3, July 1980, pp. 294-301.

Fournier, A., and Fussell, D. "Stochastic Modeling in Computer Graphics," Siggraph '80, Special Issue, July 1980.

Gibson, J.J., The Perception of the Visual World. Cambridge, England: Riverside, 1950.

Horn, B.K.P., "Hill Shading and the Reflectance Map," Proceedings: Image Understanding Workshop, Report #SAI-80-895-WA, April 1979, pp. 79-120. Palo Alto, CA: Science Applications, Inc.; 1979.

Koenderink, J.J. and van Doorn, A.J., "Local Structure of Movement Parallax of the Plane," Journal of the Optical Society of America, Vol 66, No. 7, July 1976, pp. 717-723.

Leler, W.J., "Human Vision, Anti-Aliasing, and the Cheap 4000 Line Display," Siggraph '80, Vol. 14, No. 3, July 1980, pp. 308-313.

Mandelbrot, B.B., Fractals: Form, Chance and Dimension. San Francisco, CA: W.H. Freeman Co., 1977.

Marshall, R., Wilson, R. and Carlson, W., "Procedure Models for Generating Three-Dimensional Terrain," Siggraph '80, Special Issue, July 1980, pp. 154-162.

Needham, R.C., Edwards, B.J., Jr. and Prather, D.C., "Flight Simulation in Air-Combat Training," Defense Management Journal; 4th Quarter, 1980, pp. 18-23.

Newell, M.E., "The Utilization of Procedure Models in Digital Image Synthetic Imagery," PhD Doctoral dissertation, University of Utah, 1975.

Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics (2nd Ed.). New York: McGraw-Hill, 1979.

Regan, D. and Beverly, K.I., "Looming Detectors in Human Visual Pathway," Visual Research, Vol. 18, No. 4, 1978, pp. 415-421.

Regan, D. and Beverly, K.I., "Visually Guided Locomotion: Psychophysical Evidence for a Neural Mechanism Sensitive to Flow Patterns," Science, 205, 1979, pp. 311-313.

Rogers D.F. and Adams, J.A., Mathematical Elements for Computer Graphics. New York: McGraw-Hill Book Co., 1976.

Rosenfeld, A. and Kak, A.C., Digital Picture Processing. New York, NY: Academic Press, 1976.

Shand, W.A., "Theoretical Modeling of Low Light Level Systems," Low Light and Thermal Imaging Systems, IEEE Conference, Publication No. 124, March 1975.

Smith, A.R., "Color Gamut Transform Pairs," Computer Graphics, Vol. 12, No. 3, 1978, pp. 12-19. (1978 Siggraph Proceedings).

Soland, D., Voth, M. and Narendra, P., "Real-Time Feasibility for Generation of Nonlinear Textured Terrain," AFHRL-TR-79-27, AD-A095 070, Operations Training Division, Williams AFB, AZ, January 1981.

Stenger, T., Dungan, W. and Reynolds, R., "Computer Image Generation Texture Study" AFHRL-TR-79-2, AD-A074-019, Advanced Systems Division, Wright-Patterson AFB, OH, August 1979.

Steven, M.D. and Unsworth, M.H., "The Radiance Distribution of Clear and Overcast Skies." AIAA, 555 W. 57th Street, New York, NY, A79-44554 (Paper), 1979.

Thomson, E.C., "Developments in Low Light Level Television," IEEE Conference, Publication No. 124, March 1975.

Williams, L., "Casting Curved Shadows on Curved Surfaces," Computer Graphics, Vol. 12, No. 3, August 1978, pp. 270-274.

Zimmerlin, T.A., Suthy, G.J. and Stenger, A.J., "Data Base Descriptors for Electro-Optical Sensor Simulation," AFHRL-TR-78-86, AD-A065043, Advanced Systems Division, Wright-Patterson AFB, OH, February 1979.

Zirkind, R., An Obscuring Aerosol Dispersion Model, (Volumes I and II). McClean, VA: General Research Corporation, Operations Analysis Group, 7655 Old Springhouse Road, December 1978.

APPENDIX A

OTHER OCCULTATION TECHNIQUES

As stated in the main report, the z-buffer technique is required for the Catmull subdivision method used in surface representation. The z-buffer is required because the subdivision method produces pixels in a random order. Other surface representation techniques allow other occultation techniques, but usually there is one occultation method notably more appropriate for a given surfacing method. In other words, surface representation and occultation techniques are closely related and sometimes inseparable.

Sutherland, et al.* have shown that occultation is essentially a sorting problem. They categorize techniques according to where, when, and what is sorted. Techniques have been devised to sort in object space or image space, to sort on z, sorting on edges, patches, faces, or clusters. Most methods attempt to take advantage of some coherence. There are eight kinds of coherence:

1. Frame
2. Object
3. Face
4. Edge
5. Implied-edge
6. Scan-line
7. Area
8. Depth

For AVSS, the techniques which use scan-line coherence make some sense. Lane, et al.† describe the more promising scan-line techniques for parametrically defined surfaces. There are advantages and disadvantages to these but not on the basis of their occultation method. All the occultation methods work if they are properly paired off with the surface representation methods.

---

*I.E. Sutherland, R.F. Sproull and R.A. Schumacker, "A Characterization of Ten Hidden Surface Algorithms," ACM Computing Surveys, Vol. 6, No. 1, March 1974.

†J.M. Lane, L.C. Carpenter, T. Whitted, and J.F. Blinn, "Scan Line Methods for Displaying Parameterically Defined Surfaces," CACM, Vol. 23, No. 1, January 1980, pp. 23-24.

APPENDIX B

ALTERNATIVE TEXTURING TECHNIQUES

Three texturing techniques which were considered inappropriate for AVSS for perceptual or implementation reasons are discussed here:

1. Markov processes
2. Texture tiles
3. Random mapping

MARKOV PROCESSES

A graytone texture pattern can be generated by a so-called Markov chain. The idea is to generate a sequence of tones using the probable knowledge of some given tone occurring after some previous tone or tones. All that is needed is a few initial gray values and a set of probabilities, the latter from an arbitrary textured image obtained and analyzed beforehand.

Advantages

Storage is minimal and the sequential generation technique is fast.

Disadvantages

1. Not all textures can be approximated by Markov chains.
2. The texture defined by a Markov chain applies to only one level of detail. Markov chains for the other levels are troublesome because a given texture area may be viewed at different levels of detail simultaneously, requiring either averaging of a high-resolution chain or an intricate combination of multiple chains.
3. There is no inherent continuity in Markov synthesized textures if identical probabilities are used to generate similar textures on adjacent patches. Therefore, assigning a Markov chain to each patch will result in a patch-like appearance.
4. Markov synthesis is a serial procedure, not geared to a subdivision algorithm.

TEXTURE TILES

The most advanced study dealing with periodic textures (texture tiles) is that of Stenger, et al.* They demonstrate a texture tile methodology which blends the tiles together and eliminates any periodicities. The blending also eliminates the detail in the original image, causing a blurry, indistinct texture which no longer resembles the original.

RANDOM MAPPING

Random and periodic textures such as those of Schacter† are crude representations of terrain textures, employing sinusoids or Gaussian random fields (or both). To avoid undersampling, current systems display textures at 10 levels of detail. Such textures generally have little resemblance to the real world because of their notable periodicity.

---

*T. Stenger, W. Dungan and R. Reynolds, "Computer Image Generation Texture Study," AFHRL-TR-79-2, AD-A074 019. Wright-Patterson AFB, OH: Advanced Systems Division, August 1979.

†B. Schachter, "Long-Crested Wave Models," Computer Graphics and Image Processing, Vol. 12, No. 2, February 1980, pp. 187-200.

APPENDIX C

IMAGE SPACE VS OBJECT SPACE

One of the central considerations in AVSS was whether to subdivide patches in image or object space. In object space (the choice here), subdivision of both fractals and bicubics is scalar; that is, there is only a one-dimensional subdivision. In image space, bicubic subdivision is vector (three-dimensional), requiring more hardware in parallel. Furthermore, there is no parametric functional form for fractals as there is for bicubics. Because of this, fractal subdivision in image space has no defined methodology at this time (though one may be developed in the future). Even with the advent of image space fractal subdivision, vector subdivision of fractals will be required and fractal look-up table values will probably become non-integer.

State-of-the-art fractals can be created in object space for the purpose of generating perturbed surface normals only, ignoring the perturbed fractal elevations themselves. The normals can then be applied in image space to determine surface intensities. The result will be a smooth, bicubic surface with the appearance of a fractaled texture. In any event, image space subdivision requires finding the object space normals; that is, it requires scalar bicubic and fractal subdivision. With no perturbed elevations, three-dimensional trees, forests, mountains, clouds, etc cannot be created--a disadvantage of the image space method.

In addition to the arguments expressed above, the primary advantage of image space subdivision is that it requires only ≈ 40,000 perspective transformations per frame time, as opposed to object space subdivision, which requires ≈ 5×10^6.

On the other hand, register squares are computed off-line for object space subdivision and are recomputed each frame time for image space; that is, 48 adjacent elevations must be accessed (16 each of X_e, Y_e, Z_e) and 3 x 40,000 register squares formed each frame time. In the latter case it would be inefficient to first obtain the 120,000 register squares each frame time and then perform subdivision on their respective patches. This would require large temporary storage. More efficiently, the process would be pipelined so that as a triple of register squares was formed, subdivision of their patch would begin. This implies that patches would be projected and subdivided in screen (image) space in the same order that register squares are computed; that is, in a sequential overlapping order. Object space subdivision would be more flexible since the register squares already exist. For example, projection and subdivision could be ordered in such a way that patches closer to the observer are treated first.

Object space subdivision was chosen because it allows three-dimensional fractal texturing, because bicubic subdivision hardware is rather expensive and four times as much hardware is needed in image space (3(x,y,z) + 1 intensity), and because data management is more flexible in object space.

APPENDIX D

OTHER SURFACE REPRESENTATION ALGORITHMS

Several algorithm alternatives which were considered for AVSS, but discarded, are summarized here. To clarify comparisons, advantages and disadvantages are listed for each algorithm. In each case, the disadvantages seemed to outweigh the advantages. Catmull's original algorithm was the basis for the Honeywell-Catmull algorithm chosen for AVSS.

CATMULL ALGORITHM

Though many of the techniques of the original Catmull algorithm were incorporated into the Honeywell-Catmull algorithm, there are several important differences between the two. For convenience, these algorithms are abbreviated as CA and H-CA.

In the CA patch, subdivision is continued until only one pixel center is covered. In the H-CA patch, subdivision is halted when the patch is a certain size (pixel size). The CA determines patch surface normals by taking surface derivatives whereas the H-CA determines surface normals through a quick recursive process.

CA ADVANTAGES

1. Just as with the H-CA, bicubic subdivision allows high resolution everywhere on the screen.
2. Terrain is smoothly approximated.
3. The z-buffer provides a powerful occultation test and is not subject to the potential flaws of hidden surface algorithms.

CA DISADVANTAGES

1. Determining surface normals with derivatives is slow.
2. Stopping subdivision when a patch covers one pixel center does not ensure that the patch will be pixel size. In some cases, such patches may extend over several pixel dimensions and still cover only one pixel center. This can lead to a faceted surface appearance in some cases.
3. No provision is made for anti-aliasing, realistic texturing, or shadows.

SCAN-LINE ALGORITHMS*

Traditionally, terrain has been simulated by a collection of polygons displayed in scan-line order. Defining screen space coordinates as x going to the right, y going up, and z going into the screen, a scan-line algorithm consists of two nested loops (an x loop and a y loop). During the y loop, three-dimensional polygons are intersected with constant y planes, resulting in a set of line segments in xz. During the x loop these line segments are intersected with a ray defined by the eyepoint and a screen picture element, resulting in a set of one-dimensional points. These points are then sorted in z with the smallest z being visible.

To make this scheme practical, polygons must be sorted in y before display to avoid consulting a list of all polygons each frame time. Sometimes an x sort is made before z sorting, sometimes the reverse.

POLYGON ALGORITHMS

There are many variations of polygon scan-line algorithms.† A common element, however, is the representation of terrain surfaces by flat, polygonal surfaces.

Advantages

1. Polygon-scan line algorithms are fairly inexpensive, partly because straight lines are easy to deal with and partly because hardware already exists for polygon display.
2. Intensities for truly flat surfaces are computed only once for the entire surface. To approximate a curved surface, intensity interpolation can be employed.
3. Mapped textures can be easily applied.
4. Perspective transformations are minimized because only polygon vertexes require a transformation. Screen x, y, and z polygon coordinates may be generated incrementally.
5. Anti-aliasing is easier to apply because all surface elements for a pixel are available at one time, rather than randomly with a z-buffer.

---

*Polygon, Blin, Whitted, Lane-Carpenter, Clark, Biquadratic

†I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden Surface Algorithms," ACM Computing Surveys, Vol. 6, No. 1, March 1974.

Disadvantages

1. Silhouettes are composed of straight lines.
2. Gourand shading; that is, the interpolation of intensities, allows a Mach band effect. The more sophisticated interpolation of vector normals (Phong shading* eliminates this effect but is more expensive.
3. All polygons representing the terrain must be stored. Compared to the information in bicubic patch register squares, the polygon data base is not compact.
4. Polygon algorithms depend largely on texture mapping. This technique requires large data storage, has a finite resolution, and is only two-dimensional.

   If fractals were applied to polygons, a subdivision process would by definition be required. But if fractal subdivision were used, then bicubic subdivision of curved terrain surfaces could just as well be implemented in parallel, with little extra cost and the benefit of added realism. Note that as fractals were applied to polygonal surface, some interpolation of intensities would probably still be necessary to avoid Mach bands, whereas with a curved bicubic surface under the fractal perturbations, no interpolation would be necessary.
5. The size of terrain-representative polygons is limited both by storage and sorting constraints.
6. Polygon sorting algorithms are occasionally subject to ambiguities and singularities.

BLINN ALGORITHM

This algorithm scans curved, parametrically-defined patches rather than polygons. As in a polygon scheme, a y sort is necessary for efficiency. In a polygon scheme, polygons are sorted according to the highest y value on the polygon. This is inevitably the y value of its highest vertex. Thus, only the polygon vertexes need to be considered. Blinn† attempts a sort according to the maximum y value of a curved patch.

---

*B.T. Phong, "Illumination for Computer-Generated Pictures," CACM, Vol. 18, No. 6, June 1975, pp. 311-317.

†J.F. Blinn, "Computer Display of Curved Surfaces," Doctoral dissertation, University of Utah, December 1978.

Advantages

1. Curved surfaces are displayed, adding more realism.
2. All the efficiency advantages of scan line algorithms listed under polygon algorithm advantages hold.

Disadvantages

1. Because all parts of a curved patch may potentially be maxima, the maxima tests are quite difficult to implement. Generally, iterative techniques are employed, but even these fail or are ambiguous when dealing with saddle points, cusps, and non-monotonic curves and silhouettes. Consequently, when dealing with the generation of the real world, no guarantees can be made as to the accuracy of the simulation.
2. Fractal texture and shadow algorithms cannot easily be applied since they involve a subdivision methodology.

WHITTED ALGORITHM

Whitted* avoids some of the difficulties in Blinn's algorithm by approximating a curved (bicubic) patch by the four patch edges. Furthermore, cubic curves are generated as approximations to silhouettes whenever necessary (see Figure D-1).

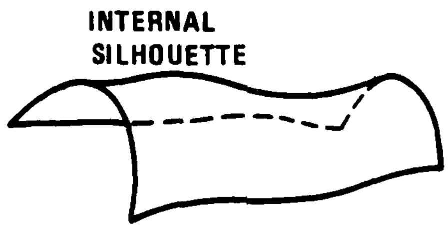


Figure D-1. Whitted approximates internal silhouettes with cubic curves.

---

*J.T. Whitted, "A Scan Line Algorithm for Computer Display of Curved Surfaces," Proceedings, 5th Conference on Computer Graphics and Interactive Techniques, Atlanta, GA, 1978.

Advantages

1. In addition to scan line advantages, patch edge maxima are easier to determine than the internal maxima in Blinn's algorithm.
2. Shadow silhouettes may also be determined by this technique.

Disadvantages

1. Since the generation of a cubic silhouette is inaccurate for highly curved patches, a curvature test is necessary.
2. As in the Blinn algorithm, an iteration technique is used to determine silhouette endpoints, and necessarily suffers from singularity problems. Some of the conditions under which the Whitted algorithm fails are illustrated in Figure D-2.

LANE-CARPENTER ALGORITHM

This algorithm is a combination of scan-line and subdivision techniques. In it, patches are sorted by their maximum possible y values. At each scan line, patches are subdivided until no subpatch overlaps the scan line (that is, when the subpatch is approximately pixel size). Subdivision can also be halted if the patch is within some set tolerance of being a planar polygon. In either case, when subdivision is stopped, the patch will be treated as a polygon and processed with a polygon scan-line algorithm.

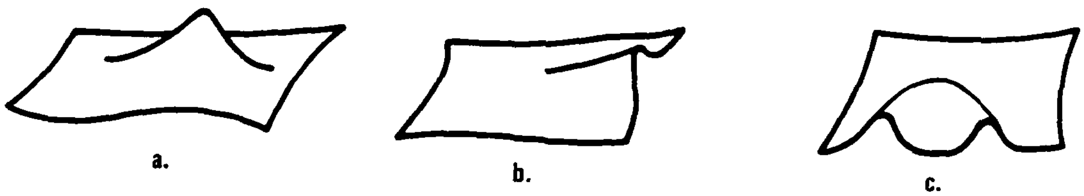


Figure D-2. Three examples of silhouettes which the Whitted algorithm does not treat.

Advantage

The advantage of this algorithm lies in its ability to minimize the number of subdivisions by making a quick flatness test. Although "holes" sometimes can appear, the Lane-Carpenter algorithm*† is quite successful when used to generate simple curved objects at a fairly constant distance.

Disadvantages

1. Under worst-case conditions (low sun angle and bumpy terrain), most patches will require subdivision down to a fine resolution. Thus the flatness test will be a liability.

2. Suppose a large, flat area is approximated by a single polygon. This inherently disallows the construction of any texture with the polygon. In other words, a flatness test is useful only when no texture is present.

3. The maximum possible y value of a patch is viewpoint-dependent and so must be located each frame time. The method by which y maximum is located for a patch involves taking many second derivatives, which is a tremendous liability for real-time simulations. An enclosing box is quicker but leads to sorting ambiguities.

BIQUADRATIC ALGORITHMS

Modeling "free-form" terrain (that is, terrain with arbitrary elevations) is best accomplished with bicubic patches. Biquadratic patches, though simpler mathematically, produce discontinuities across patch borders. Biquadratics are suitable only for a limited class of cultural features.

CUBE INTERSECTION ALGORITHM

An alternative algorithm for AVSS is the so-called cube-intersection algorithm. The idea central to this algorithm is the creation and storage of the entire mission data base off-line in a tree structure of cubes. The largest cubes would be 400 meters on a side. Each 400-meter cube would contain eight 200-meter cubes, each 200-meter cube eight 100-meter cubes, and so forth.

---

*J.M. Lane and L.C Carpenter, "A Generalized Scan-Line Algorithm for the Computer Display of Parametrically-Defined Surfaces," Computer Graphics and Image Proceeding, Vol. 11, 1979, pp. 290-297.

†J.M. Lane, L.C. Carpenter, T. Whitted, and J.F. Blinn, "Scan Line Methods for Displaying Parametrically-Defined Surfaces." CACM, Vol. 23, No. 1, January 1980, pp. 23-34.

All cubes are either "empty" or "non-empty." If no object (or no part of an object) falls within a cube, it is given an empty designation. If a cube contains part of the terrain surface or part or whole of a cultural object, the cube is non-empty.

The algorithm provides for the off-line generation of a bicubic-fractaled surface on which the cube tree structure can be superimposed. Off-line subdivision of the terrain surface results in a collection of small (2-meter) subpatches. If any one subpatch vertex falls within a cube, the cube is non-empty.

Thus, the procedure for off-line generation of a terrain surface described in a cube tree structure would be:

1. Determine bicubic and fractal coefficients.
2. Subdivide the entire terrain surface down to a 2-meter resolution.
3. Superimpose a cube tree structure over the whole mission data base.
4. Determine all empty and non-empty 2-meter size cubes.
5. Calculate diffuse intensities for all 2-meter patches and assign these intensities to the 2-meter size cubes.
6. Average cube intensities and assign to larger cubes higher up the tree.
7. Designate as non-empty any larger cube containing a non-empty cube.

In the end of the off-line preparation, all the mission data are stored in optical discs and are ready for real-time access. All textures, shadows, cultural objects, etc have been assigned in the off-line generation.

The real-time accessing of these data is accomplished with a ray tracing technique. For each pixel on the display, a ray projects outward from the pilot and intersects cubes.

1. If a ray intersects an empty cube, it proceeds to the next cube.
2. If a ray intersects a non-empty cube, the eight sub-cubes are accessed and those intersected by the ray are tested for emptiness or non-emptiness.
3. At the intersection of a ray and an opaque cube, the cube intensity is written into the pixel, and the next ray is generated. An opaque cube is a non-empty cube no smaller than pixel dimensions; that is, cube opacity is a function of pilot-terrain distance, and is used to avoid having all rays proceed to the 2-meter resolution.

Advantages

1. Because rays stop at the first surface of intersection, no hidden surfaces are dealt with.
2. Recursive perspective transformations and subdivisions are avoided; that is, no tests for patch size in screen space are necessary.
3. Flat or spherical screens may be used quite easily. In the spherical case, rays are generated so that all pixels subtend equal angles from the viewer's eye.
4. The rays are generated in scan-line order. Scan-line hardware is generally faster and cheaper than random access hardware.
5. No clipping is required since the only data considered are those data struck by a ray.
6. Anti-aliasing may be implemented by simply having several rays per pixel and averaging the intensities of the surfaces (cubes) they strike. Since an enclosing rectangle is avoided, small sampling errors are avoided.
7. Z-buffer memory requirements are minimal because the pixels are assigned intensities in scan-line order. As soon as one row of pixels is filled, the data may be read out for display and a single-row z-buffer filled again.
8. Since most of the terrain is modeled by diffuse reflection and the diffuse intensities have been calculated off-line, virtually no intensity calculations are required during real time.

Disadvantages

1. Ray intersection calculations are time-consuming and the tree search is difficult to implement. A typical ray requires six divides, one multiply, and roughly 100 adds. Although several rays can be computed in parallel, each ray must search through a tree structure of cubes in a way that cannot be predetermined.
2. Rays which project at a slight angle with respect to the horizontal may travel through a large number of cubes before intersecting an opaque cube. In such cases, on the order of 300 add operations during the tree search may be required.
3. If terrain is approximated by cubes of pixel size, staircases will appear along silhouette. Subpixel cubes would then seem necessary.

4. With a smallest cube size of 2 meters, nearby objects during low-level flight will appear unrealistic and blocky, besides enhancing staircase jaggies because of the predominance of straight line segments.

5. The mission data base and high-speed local memories must be enormous. A 2-meter resolution 48 x 800 km mission data base would require approximately 10^{12} bits. The local circular high-speed memory needed each frame time would contain roughly 10^{10} bits. This is approximately two orders of magnitude larger than the circular high-speed memory employed in the Catmull-Honeywell algorithm. Each frame time, a certain number of cubes must be transferred from disc to high-speed memory, equal to 10^6 bits/frame time. This data transfer rate is quite high and difficult to implement. In the Catmull algorithm, only a few thousand bits are transferred each frame time.

All of the above disadvantages can be reduced somewhat by implementing a "rings of resolution" structure which places the highest detail near the pilot and lower detail further away. This structure would decrease storage and data transfer rate requirements but would add some complexity. Because the disadvantages appear to outweigh the advantages, the cube algorithm will not be used.

PERSPECTIVE-STORAGE ALGORITHM

Rather than store all mission data in great detail in cubes, it could be stored as patches and subpatches. Thus, the whole data base would consist of 2-meter size patches, stored in a tree structure. When needed, a branch containing a given resolution patch would be projected to screen space. Thus, rings surrounding the pilot (and moving with the pilot) would be used to access terrain patches at some adequate (pixel) resolution. A circular fast memory and disc mission memory would be necessary. Both would be as large as the memories in the cube intersection algorithm.

Advantages

1. No subdivision is needed in real time.

2. No intensity calculation (except for an atmospheric factor) is needed in real time.

Disadvantages

1. Compared to the suggested Catmull-Honeywell algorithm, this method requires more perspective transformations per frame time because each ring of resolution must transform one patch size to the screen and since the closest patches in the rings must be no larger than pixel size, all more distant patches in a ring will be projected at subpixel size.

2. Cube algorithm disadvantages 4 and 5 hold for this algorithm.

Because of these disadvantages and because no z-buffer or anti-aliasing problems are solved, the perspective-storage algorithm is not an optimal algorithm.

CURVATURE TEST ALGORITHM

To avoid excessive storage and save on certain calculations, terrain patches could be subdivided off-line until their curvature is such that the approximation of the subpatches as polygons does not leave a fractaled appearance. Then only polygons will be stored, not register squares. During real time, bicubic subdivision is avoided, and only fractal subdivision is applied. Perspective transformation, fractal subdivision, and rectangle tests are conducted as usual.

Advantages

1. Less storage is required in both disc and high-speed memories than with a 2-meter resolution everywhere.
2. Less data are transferred from discs to high-speed memory each frame time (compared to the perspective-storage algorithm).
3. Real-time subdivision is fractal only; the absence of bicubic subdivision decreases hardware costs and computation time.
4. Because subdivision occurs during real time, full resolution is obtained for all parts of the terrain displayed on the screen.

Disadvantages

1. Under best-case conditions, the terrain will be essentially flat and the sun at a high altitude. Under worst-case conditions, the terrain will be hilly or mountainous and the sun will be at a low altitude. In such a case, many patches will have to be off-line subdivided very finely (≈2 meters). Thus, the situation goes back to very high data storage and transfer problems. Since this worst case cannot be effectively dealt with, the algorithm should not be implemented.

APPENDIX E

DERIVATION OF THE NUMBER OF BICUBIC SUBDIVISIONS

Each terrain or cultural object surface patch that falls within the observer's field of view (FOV) is projected to his/her screen and subdivided until each subpatch is no larger than a pixel. The number of subdivisions required depends on how the terrain projects to the screen, that is, it depends on terrain roughness and orientation.

A terrain model is described below in which roughness is quantified. In addition, equations are derived for the projected areas of single patches and their distributions. These equations are combined to provide the total projected area of terrain hidden and visible surfaces on the screen. From this, the number of bicubic subdivisions per frame in a "moderately bad" case is estimated.

The first step is to derive a key relationship between the orientation of a patch, relative position of the observer, and the projected area of a patch at the screen. Much of the further analysis is based on this relationship.

The imaging geometry is shown in Figure E-1. The patch is approximately at a distance η from the observer (who is in the ζ - η plane). The observer is at height ζ relative to the patch. The normal to the patch makes an angle θ with the vertical, and its projection on the $\zeta\eta$ plane makes an angle ϕ with the ξ axis (cylindrical notation). The line-of-sight makes an angle β with the horizontal. The patch size is h x h.

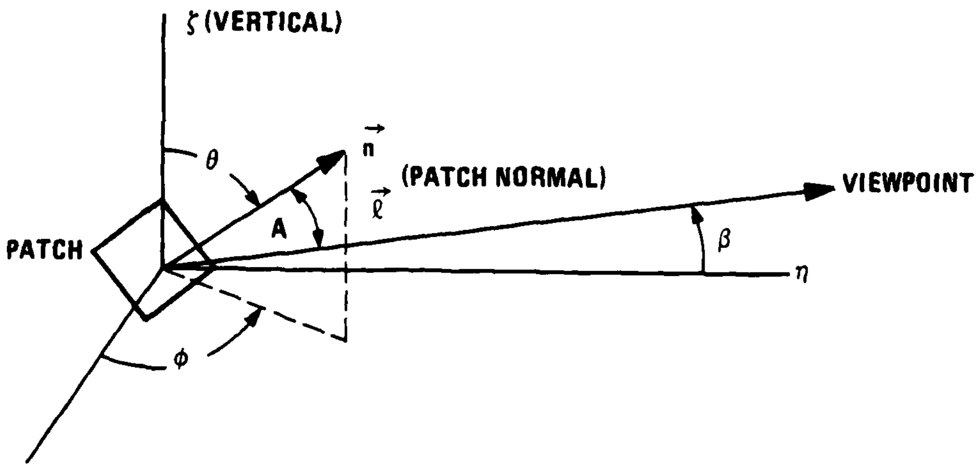


Figure E-1. Projected area of a patch at the screen.

For a flat terrain, the projected area of the patch in angular subtense at the screen is given by:

$$\text{Area} = \frac{h^2}{\eta^2} \cos A \ \ rad^2 \tag{E-1}$$

where A is the angle made by the line-of-sight vector $\vec{\ell}$ with the patch normal $\vec{n}$.

Now, $\cos A = \vec{\ell} \cdot \vec{n}$ where:

$$\vec{\ell} = [\cos \beta, 0, \sin \beta]$$

and $\vec{n} = [\sin \Theta \cos \phi, \sin \Theta \sin \phi, \cos \Theta]^T$

Then:

$$\cos A = [\sin \beta \cos \Theta + \cos \beta \sin \Theta \cos \phi].$$

Substituting (-2 into -1):

$$\text{Area} = \frac{h^2}{\eta^2} [\sin \beta \cos \Theta + \cos \beta \cos \phi] \tag{E-3}$$

For small β, the observation angle can be approximated by:

$$\sin \beta \simeq \zeta/\eta$$

$$\cos \beta \simeq 1$$

Similarly, $\sin \Theta \simeq \Theta$ and $\cos \Theta \simeq 1$ for small Θ.

Hence the area of the patch becomes:

$$\text{Area} \simeq \frac{h^2}{\eta^2} \left[\frac{\zeta}{\eta} + \Theta \cos \phi\right] (\text{rad})^2 \tag{E-4}$$

Equation (E-4) gives the projected area of a patch at a specific tilt orientation (Θ, ϕ), distance η, and relative height ζ from the observer. In practice, of course, patches can have different tilts Θ, orientations ϕ, and heights ζ with respect to the observer. Hence, to find the average projected area of a patch at a given distance, statistical models of the parameters Θ, ϕ, and ζ have to be generated. This statistical model gives rise to a means of characterizing the terrain roughness.

TERRAIN ROUGHNESS MODEL

The terrain is composed of square patches (of side h) each with an angle Θ to the vertical, an angle ϕ to the vertical plane containing the patch and the observer, and height ζ below the observer (Figure E-1). Note that Θ denotes whether a patch is tilted and ϕ

determines whether it is tilted toward or away from the observer. The following assumptions can be made about the distribution of Θ, ϕ and ζ over the entire FOV:

- o Θ, ϕ and ζ are independent random variables
- o Θ, ϕ and ζ are stationary processes in the FOV; that is, the distribution of the parameters is not a function of the location

It can be further assumed that ϕ is uniformly distributed between 0 and 2π. This means it is equally likely that the observer is looking at a patch from any direction of the compass. The distribution of Θ, the tilt of a patch to the vertical, need not be completely specified. It is completely characterized for purposes of this analysis by its mean value Θ_m. Finally, ζ is assumed to be a normal random variable with mean $m\zeta$ and standard deviation $\sigma\zeta$. $m\zeta$ and $\sigma\zeta$ are measured over the significant region of the FOV.

The terrain roughness is then completely characterized by Θ_m, the average slope of a patch, and $\sigma\zeta$, the standard deviation of the patch elevations over a small area. Figures E-2 through E-5 provide empirical data showing the distribution of tilts and altitudes of representative terrain.

THE AVERAGE AREA OF A PATCH AT DISTANCE X

From the terrain roughness model, the average area of a patch in Equation (E-4) is computed by integrating over the distributions of Θ, ϕ, and ζ. Since these parameters are assumed to be independent, the order of integration is immaterial.

The result in Equation (E-4) is first averaged over all possible orientations of the observer with respect to the patch. Since each orientation ϕ is equally likely, the probability density of ϕ is:

$$p(\phi) = \begin{cases} \frac{1}{2\pi} & 0 < \phi < 2\pi \\ 0 & \text{elsewhere} \end{cases} \tag{E-5}$$

Since the "back" of a patch must be subdivided if it is visible, negative areas (of a patch facing away from observer) should be counted as positive areas. Therefore, the average over ϕ should be written:

$$E_\phi \text{ [area]} = \frac{h^2}{\eta^2} E \left| \frac{\zeta}{\eta} + \Theta \cos \phi \right| \tag{E-6}$$

where $|\bullet|$ denotes absolute value and E [ ] denotes expected value or average.

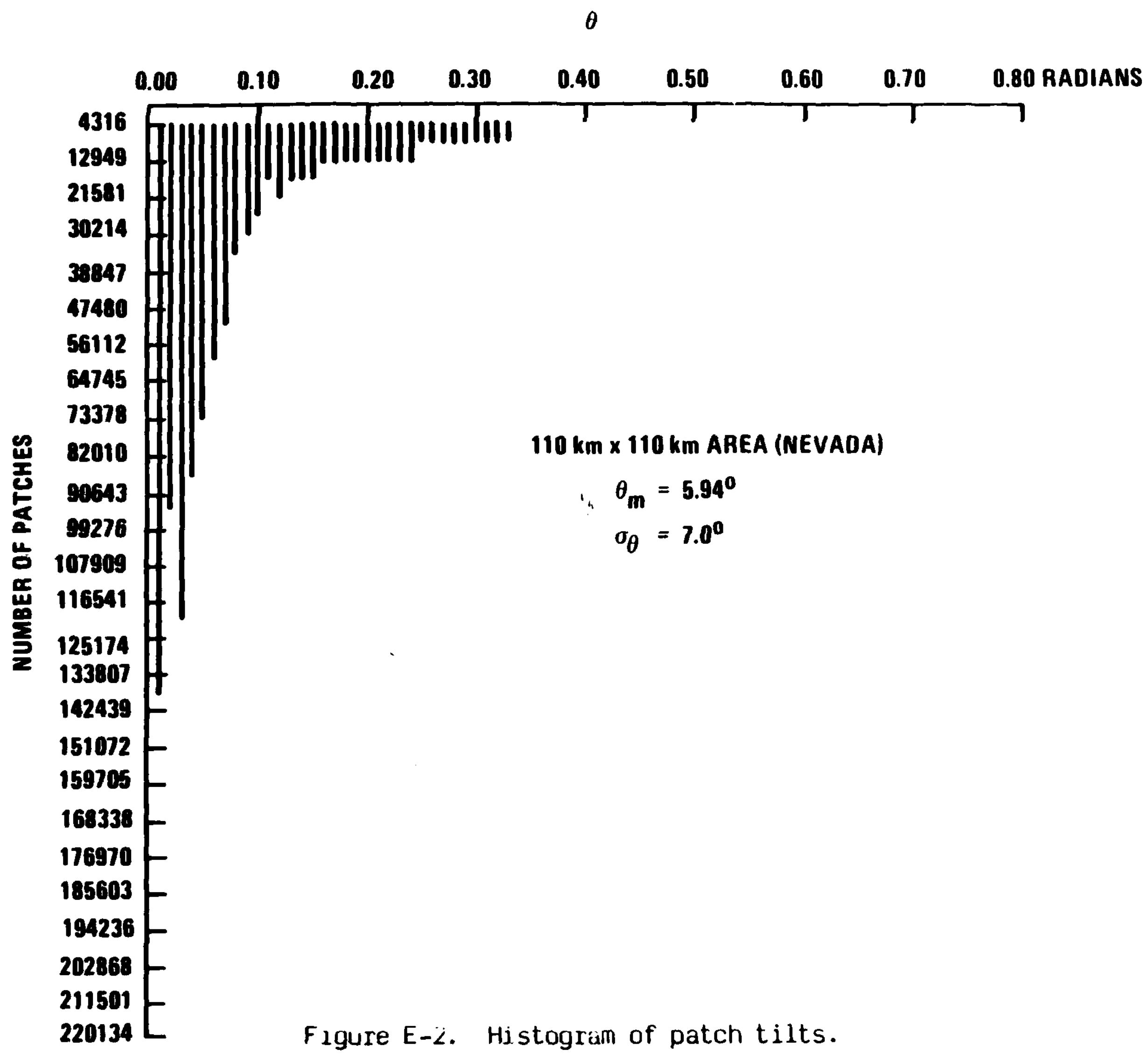


Figure E-2. Histogram of patch tilts.

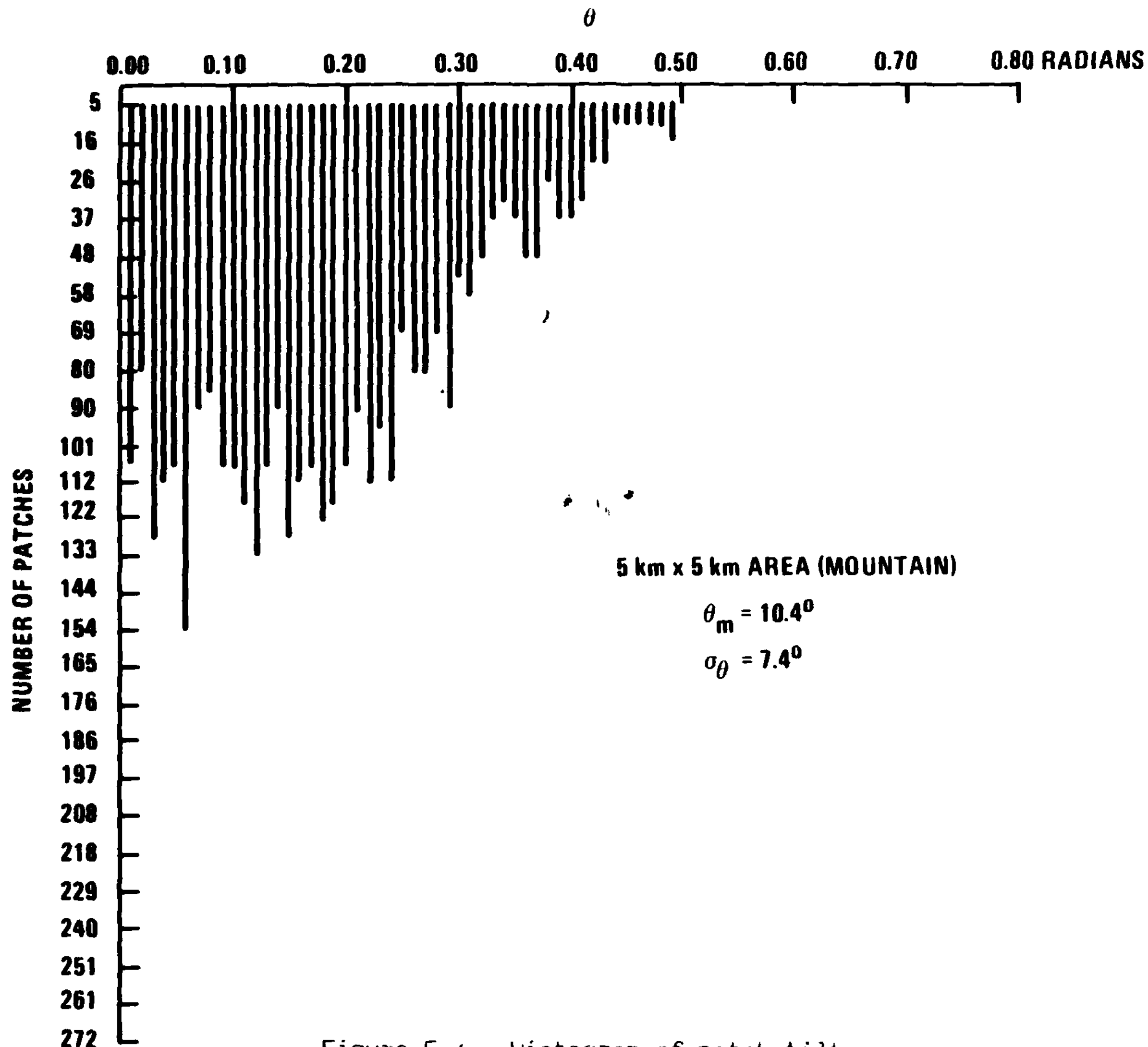


Figure E-ͻ. Histogram of patch tilts.

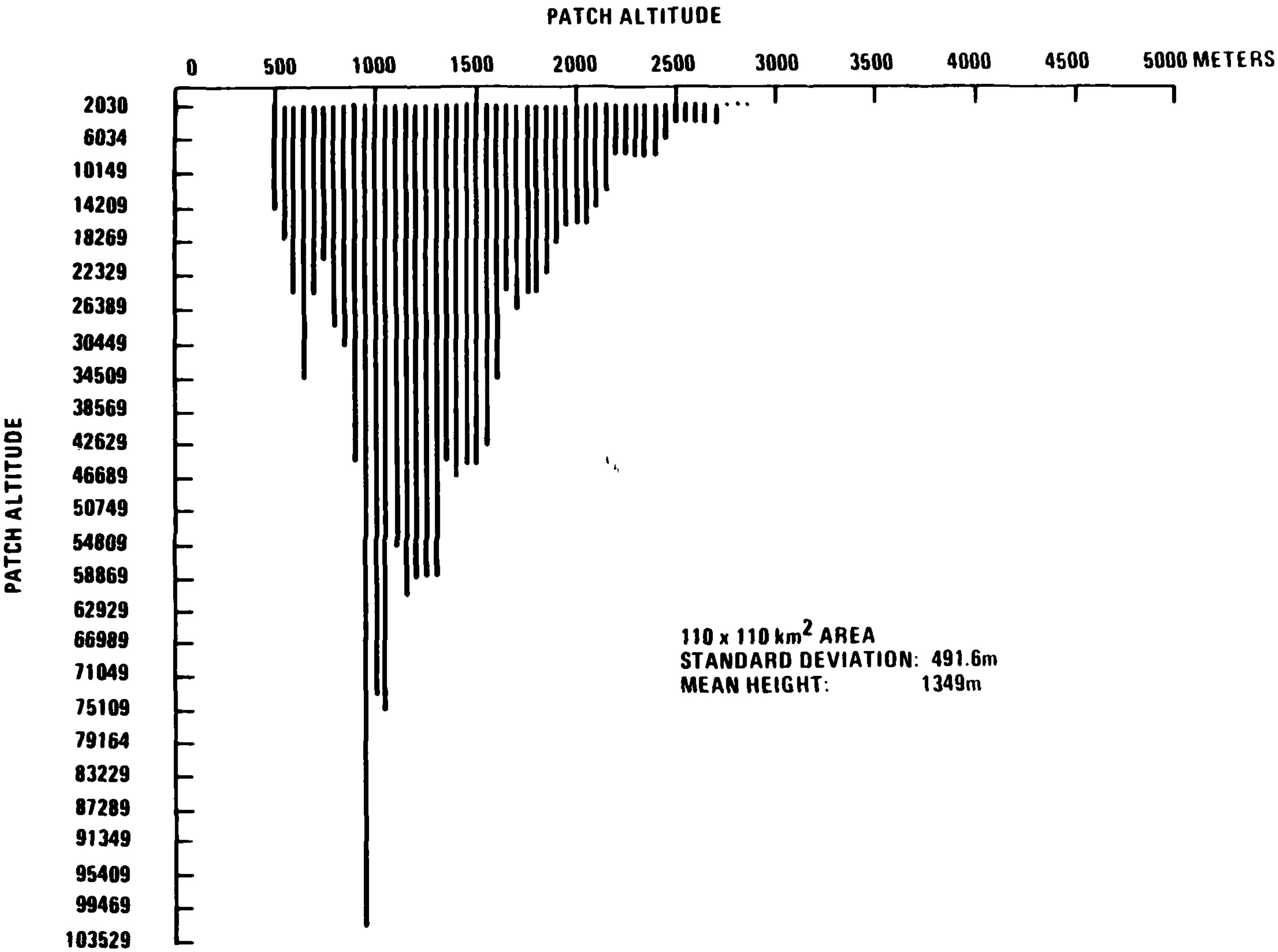


Figure E-4. Histogram of patch altitudes.

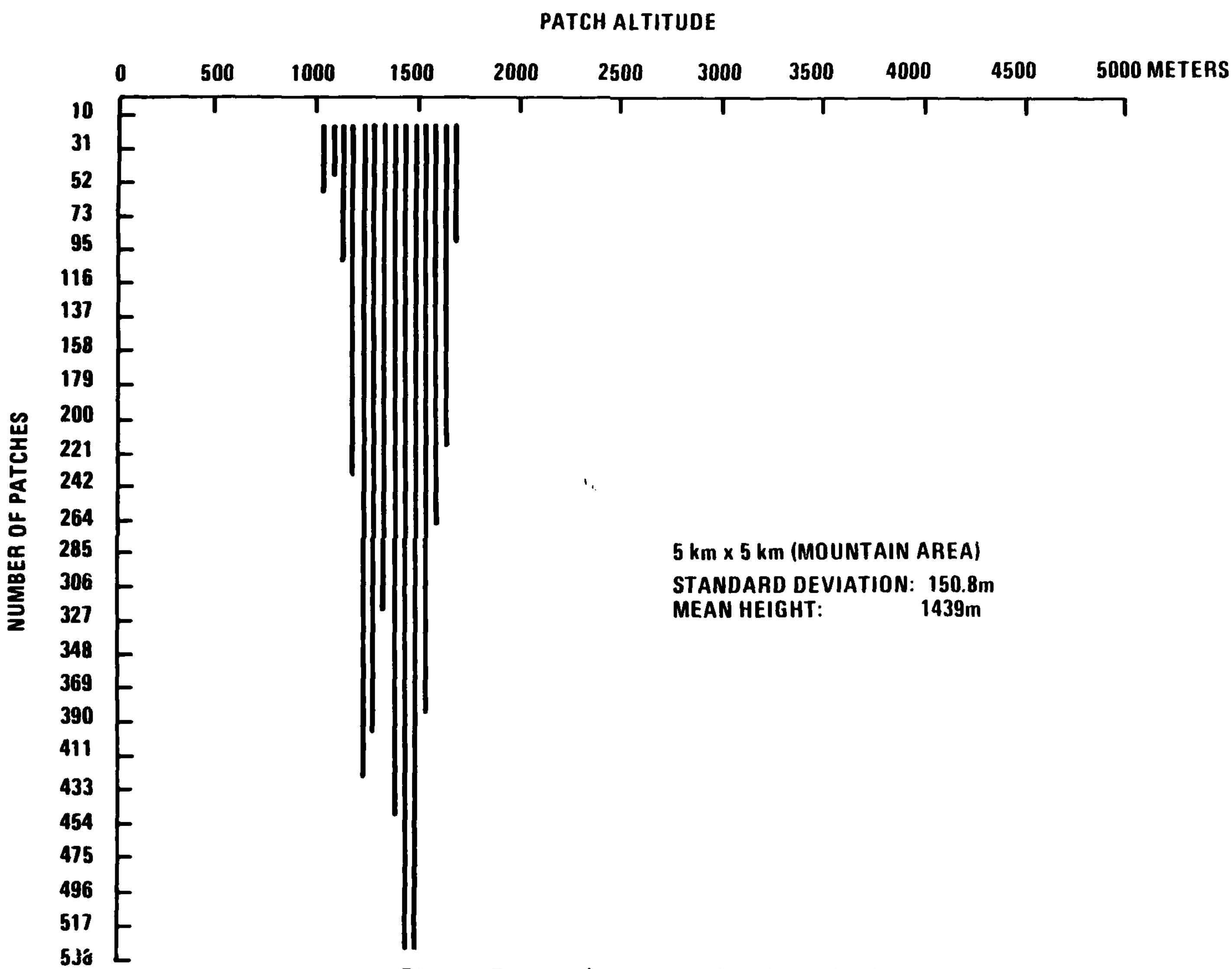


Figure E-5. Histogram of patch altitudes

Note that as ϕ varies from 0 to π, cos ϕ takes on values from between +1 and -1. Therefore, the above quantity is difficult to evaluate in closed form unless the specific value of ζ/η is known. Instead, Schwartz's inequality gives:

$$E_\phi \,[\text{area}] \leq \frac{h^2}{\eta^2} \left[\left| \frac{\zeta}{\eta} \right| + E_\phi \,[\Theta \cos \phi] \right] \qquad \text{(E-7)}$$

which gives an upper bound (conservative estimate) on the area.

Now: $$\Theta\, E_\phi \,[\cos \phi] = \Theta \cdot \frac{2}{2\pi} \int_{-\pi/2}^{\pi/2} \cos \phi \; d\phi \qquad \text{(E-8)}$$

$$= \frac{2}{\pi}\,\Theta$$

Hence: $$E_\phi \,[\text{area}] = \frac{h^2}{\eta^3} \,|\zeta| + \frac{h^2}{\eta^2} \cdot \frac{2}{\pi}\, \Theta^2 (\text{rad}\,) \qquad \text{(E-9)}$$

Since this equation is linear in Θ, averaging over all possible Θ and assuming Θ has a mean value of Θ_m gives:

$$E_\phi \,[\text{area}] = \frac{h^2}{\eta^3} \,|\zeta| + \frac{h^2}{\eta^2} \cdot \frac{2}{\pi}\, \Theta_m (\text{rad}^2) = A \qquad \text{(E-10)}$$

Note that the first term in the above equation corresponds to the flat earth case. The second term modifies the result for a rough terrain.

DISTRIBUTION OF PATCH PROJECTED AREAS

Equation (E-10) gave the average projected area of a patch at a distance η and relative height ζ from the sensor. To compute the total projected area at the screen due to all the patches in the FOV, the distribution of the patch distances must be determined; that is, how many patches are at a given distance from the observer. The number of patches in the strip $d\eta$ at a distance η is given by:

$$p(\eta)d\eta = \frac{g}{h^2}\, \psi_H \; d\eta \qquad \text{(E-11)}$$

The total projected area is obtained by integrating (E-10) and (E-11) over the near and far distances in the FOV, that is:

$$A(\ell_1, \ell_2) = \int_{\ell_1}^{\ell_2} A(\eta) p(\eta) d\eta$$

$$= \int_{\ell_1}^{\ell_2} \left[\frac{\zeta}{\eta} \cdot \psi_H + \frac{2}{\pi} \Theta_m \frac{\psi_H}{\eta} d\eta \right]$$

$$= \psi_H \frac{\ell_2 - \ell_1}{\ell_1 \ell_2} + \frac{2}{\pi} \Theta_m \psi_H \text{Ln} \frac{\ell_2}{\ell_1} \; (\text{rad}^2) \qquad \text{(E-12)}$$

Because $\ell_2 >> \ell_1$, Equation (E-12) can be written as:

$$A(\ell_1, \ell_2) = \frac{\zeta}{\ell_1} \psi_H + \frac{2}{\pi} \Theta_m \psi_H \text{Ln} \left(\frac{\ell_2}{\ell_1} \right) \qquad \text{(E-13)}$$

The projected area is, of course, a function of the near distance ℓ_1 (the distance to the bottom of the FOV). The worst case is when the horizon is at the top of the screen, as shown in Figure E-6. Then the vertical field of view determines ℓ_1 because from Figure E-6, $\psi_V = \zeta/\ell_1$.

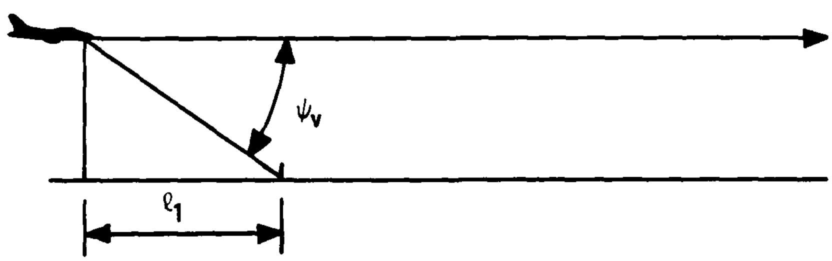


Figure E-6. Imaging geometry for the worst case.

Then Equation (E-13) becomes:

$$A = \psi_V \psi_H \left[1 + \frac{2}{\pi} \cdot \frac{\Theta_m}{\psi_V} \cdot \mathrm{Ln}\left(\frac{\ell_2}{\ell_1}\right)\right] \qquad \text{(E-14)}$$

Assuming an n x n element raster over the FOV ($\psi_V \cdot \psi_H$), the area in number of raster elements is given by:

$$A = n^2 \left[1 + \frac{2}{\pi} \cdot \frac{\Theta_m}{\psi_V} \cdot \mathrm{Ln}\left(\frac{\ell_2}{\ell_1}\right)\right] \qquad \text{(E-15)}$$

Equation (E-15) explicitly shows the total projected area for a rough terrain when the entire FOV is covered with the ground patches. In the flat earth case, it is obvious that $A = n^2$, which is intuitively correct. With rough terrain ($\Theta_m > 0$), because of hidden surfaces more than one point on the ground maps to the same point on the screen. This is the reason why the total projected area can be greater than n^2 raster elements. To see how rough terrain affects the quantity in Equation (E-15), assume $\ell_1 \approx 500$m, $\ell_2 = 20$ km, and a vertical FOV $\psi_V = 30^o$. Table E-1 shows the total projected area as a function of Θ_m, the roughness parameter.

Table E-1 shows that for $\Theta_m = 15$ (extremely rough mountain terrain) the total projected area is 2.27 times the corresponding result for the flat earth case because of hidden surfaces.

TABLE E-1. AREA VS Θ_m^o

Θ_m^o	A/n^2
0	1
2	1.16
5	1.39
7.5	1.59
10	1.78
12	1.94
15	2.27

NUMBER OF SUBDIVISIONS

Note that the projected area $A = n^2[2.27]$ is a worst case in two ways. First, $\Theta_m = 15^\circ$, which is quite rough. Second, the screen top edge is set to coincide with the horizon. This ensures that the projected scene contains no sky (the sky is not subdivided) but that the orientation is as nearly horizontal as possible, to maximize the number of hidden surfaces.

Empirical tests of the number of four-way subdivisions required per unit area of a typical projected patch demonstrated that there were (on the average) 0.75 subdivisions per unit area. This number includes all levels of subdivisions for a patch.

Thus, the total number of subdivisions is $0.75 \times 2.27\ n^2 \simeq 1.7 \times 10^6$/frame.

Note that each subdivision produces four subpatches, so that each frame we must deal with approximately $4 \times 1.7 \times 10^6 = 6.8 \times 10^6$ subpatches.

FILME

2-